KINGSTON UNIVERSITY

APPLICATIONS OF WIRELESS SENSOR NETWORKS. CASE STUDY; AN ALARM AND FIRE PROTECTION SYSTEM

ANTONIOS ZARENTIS

Master of Science in Networking and Data Communications

THESIS

KINGSTON UNIVERSITY

Applications of Wireless Sensor Networks. Case study; an alarm and fire protection system

Dissertation submitted
for the Degree of Master of Science in Networking and Data Communications

By
ANTONIOS ZARENTIS

SUPERVISOR
Dr. SOTIRIS MANIATIS

KINGSTON UNIVERSITY, SCHOOL OF COMPUTING AND INFORMATION SYSTEMS
TEI OF PIRAEUS, DEPARTMENTS OF ELECTRONICS AND AUTOMATION

JULY 2010

List of Contents

CHAPTER 1 - INTRODUCTION 9

1.1. AIM AND OBJECTIVES 10

1.2. DOCUMENT STRUCTURE 15

BIBLIOGRAPHY 16

CHAPTER 2: LITERATURE REVIEW 17

2.1. INTRODUCTION 17

2.2. COMMUNICATION STANDARDS 20

2.3. MIDDLEWARE 25

2.4. OPERATING SYSTEMS 34

2.5. APPLICATIONS IN WSN 35

2.6. HARDWARE PLATFORMS 38

2.7. THE X10 PROTOCOL 47

BIBLIOGRAPHY 48

CHAPTER 3. METHODOLOGY 53

3.1. INTRODUCTION 53

3.2. THE DESIGNING PHASE 53

3.2. THE WISECURITY LAYERS 58

3.3. THE WISECURITY SUBSYSTEMS 62

3.3.1. THE WSN SUBSYSTEM 64

3.3.2. THE X10 CONTROL SUBSYSTEM 65

3.3.3. THE SECURITY SUBSYSTEM 66

3.3.4. THE FIRE PROTECTION SUBSYSTEM 67

3.4. THE WISECURITY SYSTEM 67

3.5. A SECURITY AND FIRE PROTECTION SYSTEM; CASE STUDY 76

3.6. RESOURCES 79

BIBLIOGRAPHY 81

CHAPTER 4. CONCLUSION 82

APPENDIX A. USE CASE DESCRIPTION 83

APPENDIX B. FORMULAS FOR UNITS CONVERSION 107

B1. CONVERTING TEMPERATURE UNITS INTO CELSIUS (CROSSBOW, INC., 2006A) 107

B2. CONVERTING VOLTAGE UNITS INTO MVOLT 108

B3. CALCULATING BATTERY VOLTAGE (OCTAVE TECHNOLOGY, 2006) 108

B4. CALCULATING RSSI IN DBM (CROSSBOW, INC., 2006B) 108

B5. CALCULATING RSSI IN PWATT (CROSSBOW, INC., 2006B) 108

B6. REFERENCES 108

APPENDIX C. CLASS DIAGRAMS FOR THE WISECURITY SYSTEM 110

C1. THE ALARMSYSTEM CLASS 111

C2. THE DOWNLOAD CLASS 112

C3. THE UPLOAD CLASS 114

C4. THE RINGTHEALARM CLASS 115

C5. THE ALARMPERIODS CLASS 116

C6. THE FIREPERIODS CLASS 117

C7. THE CLIENTSSLSOCKET CLASS 118

C8. THE CONNECTTOSERVER CLASS 120

C9. THE CYGWINEXECUTECOMMANDS CLASS 122

C10. THE CYGWINJAVAAPPS CLASS 123

C11. THE CYGWINRUNCOMMANDS CLASS 124

C12. THE CYGWINUPLOADTOMOTE CLASS 125

C13. THE GETSYSTEMSTATUS CLASS 126

C14. THE SENSORSDATA CLASS 127

C15. THE SERIALFORWARDER CLASS 128

C16. THE X10SERVER CLASS 129

C17. THE ALLSENSES CLASS 130

C18. THE ALLSENSESSENDCLASS 131

APPENDIX D. WISECURITY – MANUAL 133

D1. BEFORE STARTING 133

D2. HOW TO USE THE WISECURITY SERVER 135

D2.1. CONFIGURE WISECURITY SERVER 135

D2.2. RUNNING THE WISECURITY SERVER 141

D2.3. RUNNING THE SECURITY AND FIRE PROTECTION SYSTEMS 144
D2.4. LOAD CONNECTIONS HISTORY 146
D2.5. LOAD THE WISECURITY LOGGER 147
D2.6. COLLECT WSN DATA 148
D3. HOW TO MANAGE NODES 151
D3.1. RUN A COMMAND 151
D3.2. RUN A JAVA APPLICATION 152
D3.3. COMPILE CODE 154
D3.4. UPLOAD AN APPLICATION 157
D4. HOW TO USE THE WISECURITY CLIENT 159
D4.1. MAKE A CONNECTION 159
D4.2. MANAGE THE SECURITY AND FIRE PROTECTION SYSTEMS 161
D4.3. MANAGE THE X10 NETWORK 162
D4.4. MANAGE THE WSN 165

List of Figures

FIGURE 1 : A WIRELESS SENSOR NETWORK ARCHITECTURE 13

FIGURE 2: A MIDDLEWARE INTERFACE ARCHITECTURE 14

FIGURE 3: A WIRELESS SENSOR NODE ARCHITECTURE 20

FIGURE 4: THE THREE ZIGBEE DEVICE TYPES (ZIGBEE, 2004) 21

FIGURE 5: THE 6LOWPAN ARCHITECTURE (6LOWPAN TESTBED, 2006) 24

FIGURE 6: A MIDDLEWARE INTERFACE ARCHITECTURE 25

FIGURE 7: THE MATÉ ARCHITECTURE (LEVIS AND CULLER, 2002) 29

FIGURE 8: A SHORT SQL - LIKE SCRIPT (FOR THE TINYDB) 30

FIGURE 9: MIRES ARCHITECTURE (SOUTO ET AL, 2004) 32

FIGURE 10: THE OCTAVEX MIDDLEWARE ARCHITECTURE (OCTAVEX™ TECHNOLOGY, 2006) 33

FIGURE 11: AVAILABLE MIDDLEWARE ARCHITECTURES 33

FIGURE 13: THE MINIATURE MOTE (HOLLAR, 1996) 39

FIGURE 12: RF MOTE WITH MULTIPLE SENSORS (HOLLAR, 2000) 39

FIGURE 15: THE CCR MOTE (BERKELEY, 2006) 40

FIGURE 14: THE LASER MOTE (HOLLER, 1996) 40

FIGURE 16: THE WEC MOTE (HOLLAR, 1996) 41

FIGURE 17: THE MICA MOTE (CROSSBOW INC., 2006A) 41

FIGURE 18: SENSOR BOARD [MTS310CA] (FOR MICA MOTES) (CROSSBOW INC., 2006A) 42

FIGURE 19: INTERFACE BOARD (FOR MICA MOTES) (CROSSBOW INC., 2006A) 42

FIGURE 20: THE MICA2 MOTE (CROSSBOW INC., 2006B) 43

FIGURE 21: THE MICAZ MOTE (CROSSBOW INC., 2006C) 43

FIGURE 22: THE MICA2DOT MOTES (CROSSBOW INC., 2006D) 44

FIGURE 23: THE "SMART DUST" CHIP (YANG, 2003) 44

FIGURE 24: THE INTEL MOTE (INTEL, 2006A) 45

FIGURE 25: THE INTEL MOTE 2 MAIN BOARD (INTEL, 2006B) 46

FIGURE 26: A COMPARISON BETWEEN THE MICA AND INTEL MOTES FAMILIES (INTEL, 2006C) 46

FIGURE 27: USE CASE DIAGRAM FOR THE WISECURITY SYSTEM I 56

FIGURE 28: USE CASE DIAGRAM FOR THE WISECURITY SYSTEM II 57

FIGURE 29: WISECURITY LAYERS STRUCTURE 59

FIGURE 30: FLOW CHART FOR FIRE SUBSYSTEM 60

FIGURE 31: FLOW CHART FOR THE SECURITY SUBSYSTEM 61

FIGURE 32: WISECURITY SUBSYSTEMS 63

FIGURE 33: THE DISSEMINATION AND COLLECTION SCHEMAS 65

FIGURE 34: THE WISECURITY MAIN SCREEN 72

FIGURE 35:WISECURITY STATUS LABELS 72

FIGURE 38: THE RS - 232 COMMUNICATION INTERFACE 76
FIGURE 37: MICA2 MODE 76
FIGURE 38: MPR400CB SENSOR BOARD 76
FIGURE 40: CM11 INTERFACE 77
FIGURE 41: A LAMP AND AN APPLIANCE MODULE 77
FIGURE 41: WISECURITY SECURITY LEVEL OPTIONS 78
FIGURE 42: X10 NETWORK MANAGEMENT UI 79

List of Tables

TABLE 1: ZIGBEE VS. BLUETOOTH (VENKAT, 2006) 21
TABLE 2: KEY CHARACTERISTICS FOR A MIDDLEWARE 28
TABLE 3: KEY CHARACTERISTICS FOR THE WISECURITY SYSTEM 70
TABLE 4: WISECURITY EQUIPMENT COST 80
TABLE 5: USE CASE: START PROTECTION SYSTEM 83
TABLE 6: USE CASE: PROTECTION SYSTEM DOESN`T START 84
TABLE 7: USE CASE: STOP PROTECTION SYSTEM 85
TABLE 8: USE CASE: RUN JAVA APPLICATION 86
TABLE 9: USE CASE: UPLOAD TO MOTE 87
TABLE 10: USE CASE: COMPILE 88
TABLE 12: USE CASE: RUN COMMANDS 88
TABLE 12: USE CASE: READ FROM THE WSN 89
TABLE 13: USE CASE: MANAGE SERVER 90
TABLE 14: USE CASE: MANAGE X10 SERVER 90
TABLE 15: USE CASE: MANAGE FIRE SUBSYSTEM 91
TABLE 16: USE CASE: MANAGE EMAIL 92
TABLE 17: USE CASE: MANAGE SECURITY SUBSYSTEM 93
TABLE 18: USE CASE: START LOCAL SERVER 93
TABLE 19: USE CASE: STOP LOCAL SERVER 93
TABLE 20: USE CASE: MANAGE VALIDATION 94
TABLE 21: USE CASE: NON VALID REGISTRATION KEY 95
TABLE 22: USE CASE: TURN AN INDIVIDUAL X10 DEVICE ON 96
TABLE 23: USE CASE: TURN AN INDIVIDUAL X10 DEVICE OFF 96
TABLE 24: USE CASE: TURN ALL LIGHTS OFF 97
TABLE 25: USE CASE: TURN ALL MODULES OFF 98
TABLE 26: USE CASE: TURN ALL MODULES ON 99
TABLE 27: USE CASE: TURN ON SPRINKLER 100

TABLE 28: USE CASE: TURN OFF SPRINKLER 101
TABLE 29: USE CASE: TURN ON ALARM 102
TABLE 30: USE CASE: TURN OFF ALARM 102
TABLE 31: USE CASE: START SECURITY SUBSYSTEM 103
TABLE 32: USE CASE: START FIRE SUBSYSTEM 104
TABLE 34: USE CASE: STOP SECURITY SUBSYSTEM 105
TABLE 34: USE CASE: STOP FIRE SUBSYSTEM 105
TABLE 35: USE CASE: MANAGE INTERVAL 106
TABLE 36: USE CASE: MANAGE LEDS 107

CHAPTER 1 - INTRODUCTION

The last years improvements in both telecommunications and electronics industry lead to the creation, improvement and maintenance of low cost sensor networks referred to as Wireless Sensor Networks. However, such networks suffer from a number of dysfunctions, including routing schemas, power management and application-based limitations.

A Wireless Sensor Network (WSN) normally is combined from a number of tiny nodes referred to as sensor nodes. Such nodes have the ability to provide wireless communication techniques and data gathering schemas with low power consumption architectures. A WSN may be used to gather information from its environment. Such information may include temperature, vibration, humidity, geographic location and other. A WSN architecture is implemented on an over the air communication. Such communication is either over the 2.4 GHz ISM band or the ZigBee (IEEE 802.15.4) communication protocol.

This document basically based on the unique design and application constraints of such networks. Current applications for WSN control and management, characterized as single applications. These applications eliminate the concurrent support of two or more applications, in such way that they exploit at the maximum the usage of a WSN. In addition, they are restricted on the area of data collection, something that eliminates the integration of the WSN with other networks. We expand the capabilities of a WSN, from a pure data collection network to a network that is capable to integrate an autonomous decision subsystem that reacts with an x10 – based network. Additionally, we add remote control and management of the whole system, while taking under account the constraints of a WSN. These constraints may be power consumption, narrow geographical coverage, data dissemination and data collection. Finally, a series of TinyOS – based development tools are included for managing and programming sensors hardware.

1.1. Aim and Objectives

Wireless Sensor Networks (WSN) is a subject of great interest both in academic and industry society. Applications for WSNs are currently used in agriculture, for managing environmental data such as temperature and humidity rates. In addition, vibration based senor boards are widely used in seismology for performing wave length measurements. Medicine and environment forecasting are also two newly sectors that already adopt WSNs technologies.

In order for these applications to be sufficiently maintained, integrated networking techniques are needed. Even though, there are plenty of networking methodologies available in wireless communication, they do not fit to the sensor network` s requirements. Akyildiz et al (2002) indicates out the basic differences between a common ad – hoc network and a WSN, which are the following:

- Sensor Networks can normally handle indefinite sensor node devices, in contrast to ad – hoc networks where nodes are definite.
- Sensor nodes are deployed closed to each other, as opposed to standard network nodes which can be far apart from each other.
- Failures on nodes are more common on sensor networks, due to the fact of the dynamics of the infrastructure used.
- Topology changes in a sensor network take place more often than an ad – hoc network.
- The communication methodologies in sensor networks completely differ from the ones in ad – hoc networks.
- Constraints (processor power, battery limitations, and bandwidth limitations) make sensor networks to create more complicated issues relatively to maintenance and analysis. And finally,
- The lack of header information in the communication protocols used in WSNs reduces the traffic in the cost of increasing the local processing of data in the sinks.

Normally, a WSN consists of a number of tiny devices named as sensor nodes. Sensor nodes are able to gather information from their environment, aggregate data and forward it on a central node over a wireless link. Such sensors may perform

temperature measures, vibration light measures and movement detection. Current sensor processing power is limited on few MHz and low RAM memory. The microprocessor primary scope is to manage sensed data before transmitting it to the sink. However, data can be aggregated or altered according to the application scope. The purpose is to decrease as far as possible the power needed for transition. Communication can be performed via an antenna, a cell phone connector which triggers SMS messages or via a serial interface (Haenselmann, 2005). Nodes in a WSN are closely deployed to each other. There are three phases for positioning the sensor nodes and handling topology changes: a) it is the pre – deployment phase, where nodes are massively placed (i.e. dropped from an airplane) or placed one by one, b) The post – deployment phase where a number of nodes may need to be replaced due to hardware problems and c) The re – deployment phase where new nodes can be added. In order for a WSN to be able to handle the massively placed nodes or topology changes, there is a need for support of efficient topology awareness protocols (Akyildiz et al, 2002).

Akyildiz et al (2002) discuss on a survey paper the limitations, capabilities and constraints of the WSN, against the OSI stack. The physical layer is normally responsible for power management issues, modulation techniques, propagation effects. The physical layer research issues primary focus on small size, high computational power, excess memory, and low price embedded devices. Beyond the hardware issues, key areas that the physical layer holds are the modulation - demodulation techniques, the available communication techniques (including encoding - decoding). The combination of all those have to provide low energy cost for the network. The data link layer is responsible for providing an interface to the network layer, error control, data frame detection and data frame multiplexing (Tanenbaum, 2003 and Akyildiz et al, 2002). Data link protocols basically emphasize on the power consumption of the overall network. Medium Access Control Protocols (MAC) should ensure low power consumption and fair communication between the nodes. Error detection is an integral part for the WSNs. A portion of applications require reliable transferring of information among the nodes. Error detection algorithms, again, should ensure low power consumption and on the other hand avoid overflow of the network. However, current researches are basically

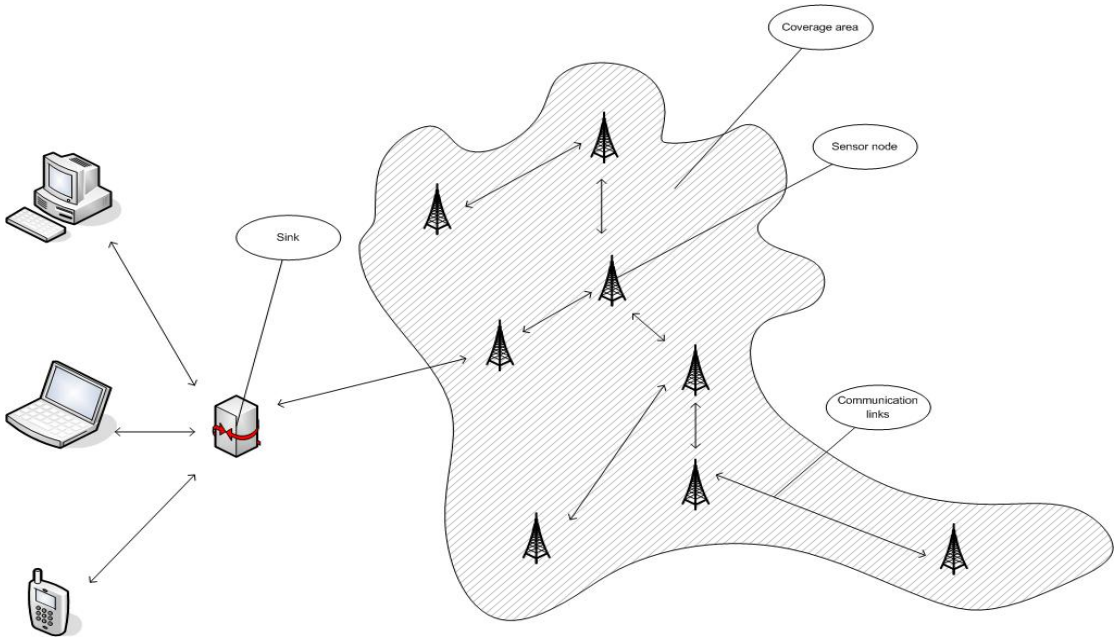
emphasized on a static WSN. Both Younis et al (2006) and Akyldiz et al. (2002) agree that a Medium Access Control (MAC) Mechanism should be based on the following key characteristics.

- **Scalability**: As a network may be constitute from hundreds of nodes, MAC protocols have to be able to provide fair communication and dejection of collisions.
- **Delay - Prediction**: A portion of WSN applications require delay prediction of data transfer. On such application the data link layer is an integral part as it has to perform data transfer scheduling and medium access arbitration.
- **Adaptability**: Corresponding to the WSN application scope, the MAC protocol architecture should be adaptive. For example in a trigger based application, data communication is performed when an event take place. In other words, there is no continuous data transfer. On the other hand, another part of WSN applications send and receive data packets continuously (for example in a query event) on their operation life. MAC protocols should adaptively provide fair medium access on both data communication circumstances.
- **Energy – efficient**: An integral part of a MAC protocol is to be energy efficient. This can be done by eliminating collisions, exploit on the maximum the frequency band and use of sleep and wakeup modes for a mode.
- **Reliable**: Even if it is not the case for all the WSN applications, reliability may be an integral part for a portion of applications. There is an analogy between the reliability and the power consumption of a WSN. The more reliable the MAC protocol is the more power is needed.

The network layer manages the routing techniques that may allow the nodes to negotiate with each other and the sink (Sink is referred as the head node of a WSN). The network layer lacks of dynamic topology changes and sensor allocation and management. The need of a transport layer for a WSN is when it is planned to be accessed from the Internet or from other networks. In the Internet, transport layer plays a major role for reliable data transfer. WSN`s transport layer could not play the same role as it actually does on the Internet. Karl and Willig (2006), notice the reasons that make transport layer different from the Internet. Firstly, reliability is performed via a combination of the layers. Moreover, a WSN is characterized from eliminated power, memory and computational power. Additionally, a WSN as a whole is formed for a specific task, thus it is possible to make a combination of protocols that serves that task and let only one protocol to interact with other protocols. The authors finalize, with the need of down – to – up protocols approach

for achieving better communication results. The application layer is the interface between the user and sensor network. The main drawback regarding the application layer is that applications already developed are single-based application. In other words, they do not provide scalability; instead they serve only the task for which they are developed. Other issues such as fault tolerance, power elimination and transmission of data techniques are of great research interest.

The WSN technology will be an integral part of life. Already hardware with sense capabilities is used in medicine. WSN play a major rule in military applications



and so on.

Figure 1 : A Wireless Sensor Network Architecture

A key factor that makes WSNs useful and characterizes them as a future trend is that they can be placed almost in any phenomenon that humans want to monitor, such phenomena include fire protection, alarm systems, surveillance monitoring and so on. Dynamic Sensor Networks (DSN) is an extension of traditional Wireless Sensor Networks that allow the monitoring of change position networks. However, DSN involve different approaches from traditional WSNs either in terms of routing and data management. In addition, routing management schemas should enforce QoS, fairly allocation of nodes and/or resources and reduce power consumption.

Schott et al. (2005) analyze the main characteristics for DSN. A key characteristic for a DSN is a self management location identification system. It should periodically monitor sensor`s position using a Location Identification Technology. Next, data gathering techniques should be implemented over low power routing protocols. In addition, authors agree that spatial addressing may reduce power consumption. Finally, the dynamic scattered nodes should trigger any update events when they change their position. The whole management and monitoring of nodes can be achieved using a middleware interface. The middleware interface is responsible to interconnect the network itself with the application (Figure 2). Middleware interfaces are implemented upon a network API (normally a Java Network Interface API) which is able to provide efficient data gathering and routing techniques.

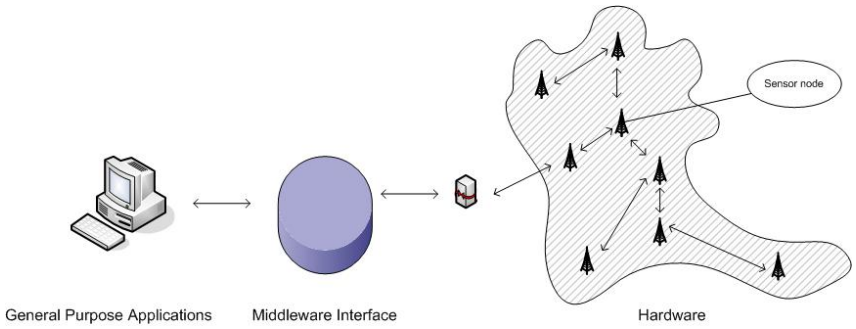


Figure 2: A middleware Interface Architecture

The same authors conclude that middleware interfaces are able to provide powerful hardware and network management but not without the existence of an operating system (OS).

The proposed dissertation is going to handle issues which are part of the application layer stack of the WSN. In particular we will try to apply a new middleware – based not application – driven framework (not focus on the application itself). The above will be part of a case study, referred as an alarm and fire protection system.

Our general scope will include:

- The deployment of a number of sensor nodes (integrating a low power mesh network) for observing and managing application's and network's oriented constraints,

- The implementation of a new application using nesC language supported by the TinyOS. Our objective is to perform surveillance observations with the use of the sensor devices.
- The development of a seamless middleware application implemented on Java between the WSN and the end user.
- An end user – based application, using Java, for network status observations including power and coverage performance, on the fly data collection and data delivery to the sensor nodes.
- A profile implementation for data acquisition and network management over the internet using Java – based techniques.
- The implementation of a seamless interface with the use of x10 – based devices for auto control electronic devices based on an auto control algorithm with the use of the surveillance observations.

1.2. Document Structure

On the second chapter, an in depth review on the current literature is presented. Initially, we focus on the data gathering techniques, the data aggregation and the communication standards for the WSNs. Then, we introduce the role of a middleware for the WSNs, the constraints and achieves made so far. Next, we analyze the available operating systems for the WSNs, including drawbacks and advantages of each one. Afterwards, we refer to the up to date available applications and the key characteristics of the application layer for a WSN. Finally, it is added a survey on the available hardware platforms, highlighting the capabilities of each hardware platform separately.

On the third chapter, it is introduced and analyzed the proposed multi – application based system, named as WiSecurity system. On this chapter, flow charts, UML diagrams, algorithms discussion and the resources analysis take place. In particular, the WiSecurity system is analyzed according to the subcomponents from which is integrated. Finally, we categorize the functional requirements while we make a focus on the most important classes of the system, by making a short description of each one.

Bibliography

[1] AKYILDIZ, I., Su, W., SANKARASUBRAMANIAM, Y., CAYIRCI, E., 2002. *A survey on Sensor Networks*. Georgia Institute of Technology. IEEE Communications Magazine.

[2] AL-KARAKI, J., KAMAL, A., 2004. *Routing techniques in wireless sensor networks: a survey*. Iowa State University. Hashemite University. IEEE Wireless Communications.

[3] HEINZELMAN, W., B., MURPHY, A., L., CARVALHO, H., S., PERILLO, M., A., 2005. *Middleware to Support Sensor Network Applications* [online]. University of Rochester.

[4] TANENBAUM, A., S., 2003. *Computer Networks*. Pearson Education International.

[5] YOUNIS, M., NADEEM, T., 2006. *Energy efficient MAC protocols for wireless sensor networks*. University of Maryland. Baltimore.

[6] KARL, H., WILLIG, A., 2005. *Protocols and Architectures for Wireless Sensor Networks*. Published by John Wiley and Sons.

[7] SCHOTT, B., PARKER, B., SRIVASTAVA, M., JONES, M., 2006. *Dynamic Sensor Networks*.

CHAPTER 2: Literature Review

2.1. Introduction

Authors Al-Karaki at al. (2004) analyze current researches regarding data gathering, aggregation and processing techniques in WSN. Such techniques lack of an efficient power consumption mechanism and communication bandwidth. In this case, even if there are data aggregation techniques they do not improve the overall network lifetime and do not allocate efficiently the available resources thorough the network.

The designing phase for WSNs has to be distinguished from the other wireless-based technologies, due to the fact that WSN techniques suffer from a number of constraints (Al-Karaki at al., 2004). Authors, Akyildiz et al (2002) agree that newly management techniques should take under consideration an efficient pattern for sensors management, task assignment on each node on the network and user queries management. Al-Karaki at al. (2004) note the need for distinguish traditional wireless networks from WSNs. A traditional wireless network may hold only a small number of peers, instead a WSN is compromised of a huge number of nodes. Moreover, a WSN implements a data – centric approach. In addition, as WSN aris limited in power, their operation should follow a low power consumption pattern. Nodes on a WSN should eliminate redundant data transmission. Such data may include handshake techniques, MAC address advertisement and so on. Due to those significant differences between traditional wireless networks and the WSNs newly research approaches should take into consideration the unique design characteristics of a WSN. Such special design characteristics are the power consumption, limited processing power and narrow bandwidth allocation. So far, many WSN-based techniques do not take under account these constraints.

A common designing key that fully characterize a WSN is that sensor nodes are usually stationary developed. Additionally, WSNs management and design process is based on the task they perform. This is the reason that many WSN are concerned as application-based networks. Akyildiz et al (2002) highlight the need of developing and manage not application driven protocols. The reason is that application-based

WSN do not take under consideration the unique constraints of both transport and network layer. The same authors also agree that positioning-based schemas improve the overall network lifetime. It is efficient for a neighbor node to know the closest to him node, in order to forward data with the minimum cost, both in terms of power and bandwidth allocation.

Al-Karaki et al. (2004), summarize that as far as the differences between sensor networks and traditional networks are eliminated, newly created algorithms should be implemented in order to manage WSN efficiently and eliminate current network constraints.

Akyldiz et al. (2002) on a survey paper, analyzing the OSI stack layer against the WSN architecture. The authors emphasize on the key factors that affect the overall network life. Additionally, they refine current open issues thorough the OSI-stack. Such factors may include:

- **Power limitation**: The goal is to increase overall network life by minimizing power consumption. The above can be achieved with the development of low power consumption architectures. It is important, that power consumption techniques are not only part of the physical layer. Instead bandwidth allocation, modulation techniques, addressing and other techniques, which served from other layers, should take into consideration WSN`s power limitations. On more important case, is the sleep off mode of nodes. Sensor nodes, which are turned off (due to errors or power exhausted) should not affect the overall network status. Data should be forwarded from the rest healthy nodes of the network.
- **Extension of power level**: This factor complements the previous one. For achieving more power for the network, nodes can be turned on a sleepy mode when there is no need of data transmission.
- **Robustness**: The whole architecture should enforce the normal operation of the network, in case of failures. Akyildiz et al (2002) refer that the fault tolerance of a node can be modelled using the Poison distribution for catching the probability of not having a failure within the time interval 0 to t.

  $$R_k(t) = e^{-\lambda} \cdot k^t,$$

  where λ_k is the failure rate of a node k.

- **Hardware Enhancements**: Four components are compromise a sensor node. (figure 3). These are the sensor itself (a hardware component which is able to sense information from its environment), a tiny processor (a hardware component which operates in low MHz, responsible to process sensor`s data and flash it to memory), the transceiver component (a hardware component enabled as receiver or transceiver using a communication channel) and the battery. In addition to the above there can be additional components. Such components vary according to the application scope. For a geographic location-identification system a GPS component could be used. In case of an environmental monitoring system power generation components are used. In order for the network to operate efficiently both the application and the network infrastructure have to provide seamless integration with all the above components. Hardware enhancements also involve reducement of hardware size, cost elimination and power enhancement.
- **Scalability**: Refers to the ability to add a number of nodes in an already deployed network, with no change of the network performance. In the most cases a WSN is composed from a great number of nodes scattered on the phenomenon. The integrated application and network schemas have to efficiently integrate all the available nodes, by utilizing a high level of density.
- **Heterogeneity**: Current WSN protocols are not taking into account interoperation with other networks. Moreover, most protocols are hardware oriented. Network-based protocols should operate over multi platform infrastructure.
- **Topology self – configurations**: Deploying nodes for the first time should be managed by an autonomous localization methodology which may be provided from the available network schemas, thus leading to a better synchronization technique.
- **Transmission media**: Current radio-based protocols support three transmission types for a node. These are the radio transmission (usually over the 2.4 ISM band), infrared and optical media transmission. The transmission decision type should be compliant to the international standards. Additionally the media transmission should provide the least propagation loss and bandwidth allocation.

- **Ethics and security issues**: WSN` s applications should be developed and deployed according to the international regulations. Newly, created communication schemas are able to apply encryption mechanism. Such schemas enforce confidentiality of data transmission.

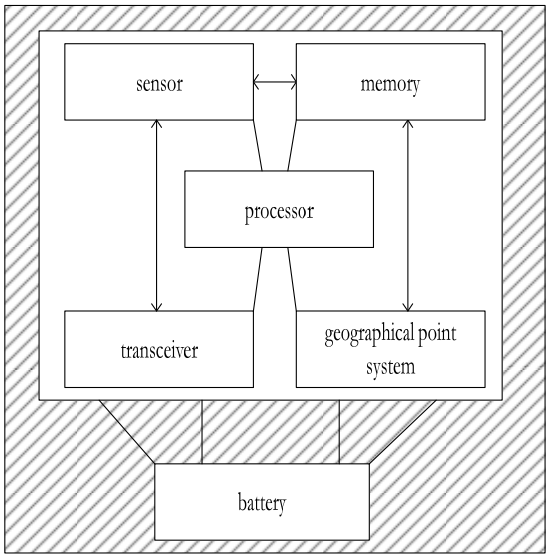


Figure 3: A wireless sensor node architecture

2.2. Communication standards

There are already considerable advancements in hardware available technologies. Such advancements lead into low power communication techniques. Innovative techniques on both nanoelectromechanical systems (NEMS) and the microelectromechanical systems (MEMS), lead to the development of low power hardware devices. However, as the authors Akyildiz et al (2003) indicate even if communication techniques are concerned as innovative they are not significantly eliminate power consumption.

The ZigBee standard is a collection of protocols that implemented upon the IEEE 802.15.4[1] standard. ZigBee, is a low power consume protocol that is able to be adopted on a variety of applications. The main characteristics for the ZigBee

[1] The IEEE working group of the IEEE 802, which focuses on the Wireless Personal Area Networks (WPAN)

standard is that enables lower power consumption, with the smallest cost and lower latency in relation to other WPAN available technologies. (Table 1). ZigBee standard operates over the following radio bands; these are the 915 MHz (used in USA/ Europe), the 868 MHz (used in Europe) and the 2.4 GHz (used in Japan). ZigBee is able to offer up to 250 kbps data transfer. (Stojmenovic, 2005).

	ZigBee(IEEE 802.15.2)	Bluetooth
Power Consumption	10 mA (≥2 years life)	100 mA (hours)
Production Cost	1.0 $	3.0 $
Development Cost	50 % less than Bluetooth	
Data rate	250 kbps	720 kbps
Number of nodes	≥ 255	7
Latency	15 ms	> 3 s
Interference	Low	High

Table 1: ZigBee vs. Bluetooth (Venkat, 2006)

ZigBee enabled devices are divided into the following three components (Figure 4). The first component is the ZigBee coordinator (ZC). ZC is the central device in a network. A ZC component primary task is to initiate the network. It operates as a coordinator on its network region.

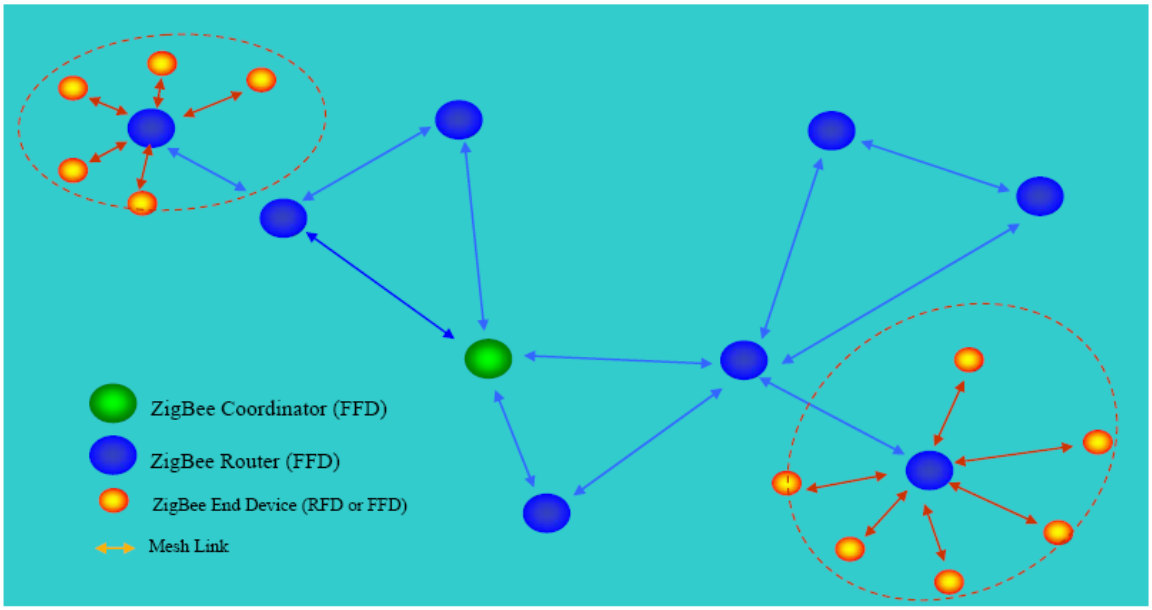


Figure 4: The three ZigBee device types (ZigBee, 2004)

This can be done either by represented itself as a master node or act as a bridge interface to other networks. The ZBC stores information related to the network. Additionally, it operates also as a repository for encryption/ security keys. The second component, part of the ZigBee standard, is the ZigBee Router (ZBR). ZBR is optional. It enables the network extension to other devices. In addition, ZBR also acts as an address allocation mechanism for the hardware devices on the network. The ZigBee End Device (ZBE) provides communication operations with a ZBC or ZBR components. It actually operates as a child node. ZBE is characterized from low memory size, power and processing consumption. In contrast, ZBC and ZBR due to its role should yarded with more power and processing capabilities.

The most basic ZigBee standard topologies are the following:

- **Star topology:** In a star topology the ZBR component actually acts as a coordinator for the whole network. A ZBR component, at least theoretically, is able to communicate concurrently with up to 65,536 ZigBee End Devices (ZigBee, 2004).
- **Cluster tree topology:** A cluster tree topology enforces network efficiency. Such topology enables P2P communication among clusters, with almost zero routing overhead. The ZigBee standard is able to operate with up to 225 clusters of 254 nodes each (ZigBee, 2004). Heile (2005) indicates that the cluster tree architecture is suitable for latency tolerant application.
- **Mesh topology:** Mesh topology enables network scalability as in this topology schema any node in the network is able to communicate with any other node on the network either using a ZBR coordinator component or a ZBE component which actually acts as a router. (ZigBee, 2004).

The ZigBee standard is basically tries to eliminate power consumption and provide high rate communication rates among the network`s nodes. ZigBee standard operates in beaconing or non – beaconing modes (Stojmenovic, 2005). A star topology normally implements the non – beacon mode. It uses NP-CMS ((Non

Persistent CSMA)) channel access mechanism. Additionally, due to small size of frames there are not presented any RTS/CTS control packets. Moreover, the backoff time in case of collision is significant reduced. A general architecture of a non – broadcasting network may be a central node (The central node is attached on an external power source) and a number of nodes (The rest of nodes are battery supplied). In contrast, beacon based network architecture is composed of one or more ZigBee Routers (ZBR). The ZBR routers actually act as an access point which transmits beacons periodically to the other network devices for association. The beacons interval ranges from 15.36 ms to 251.66 ms, a fact that enables both low duty cycle and low power consumption. Beacon based networks, use the Direct Sequence Spread Spectrum (DSS) modulation schema for radio transmission. In addition the Binary Phase-Shift Keying [2](BPSK) modulation schema is used for the 868 and the 915 frequency bands and the Quadrature Phase-Shift Keying [3](QPSK) modulation schema for the 2.4 GHz frequency band.

A great drawback for ZigBee communication standard is that is the interoperability level with existed traditional IP networks, like the Internet. Such drawbacks in conjunction to the data transfer bandwidth limitation , the Maximum Transfer Unit (MTU) and power consumption lead to a new announced communication standard, the LoWPAN (6lowpan – IPv6 over Low Power Wireless Personal Area Networks) standard issued by IETF. The 6lowpan is able to enable the interoperation of the ZigBee standard with traditional IP – based networks (Kushalnagar and Montenegro, 2006). However, the actual research group scope for 6lowpan, is to integrate IPv6 over low power WPAN. The 6lowpan working group actually aims to the IPv6 packets encapsulation over the IEEE 802.15.4 standard (Figure 5).

[2] BPSK uses two phases for modulation from the original signal.

[3] QPSK uses four phases for modulation from the original signal.

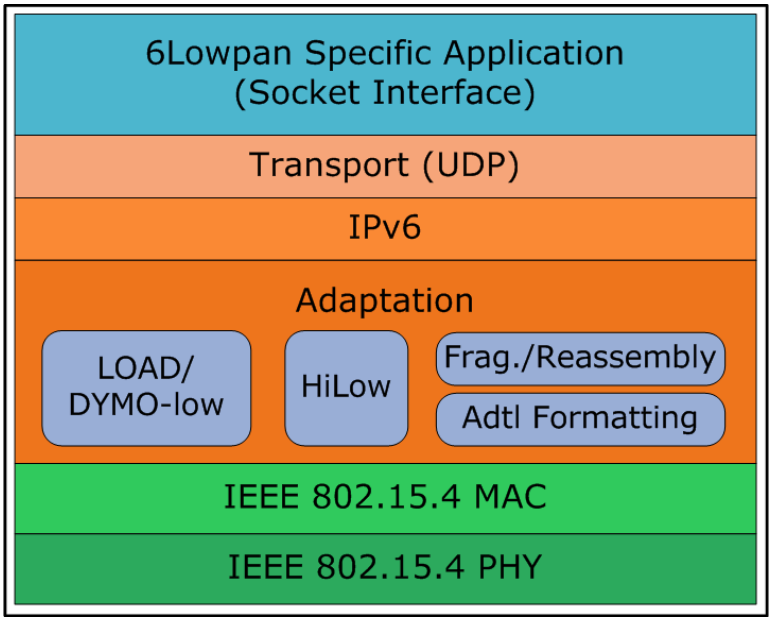


Figure 5: The 6lowpan architecture (6lowpan Testbed, 2006)

The 6lowpan standard inherits most of the characteristics of the IEEE 802.15.4 standard. Authors, Kushalnagar and Montenegro (2006) list the basic characteristics of 6lowpan below.

- **Small packet size**: Packet size on 6lowpan is 81 octets for transmission.
- **Media Access control addressing**: 6lowpan implements either 16 or 64 – bit long for MAC – Media Access Addressing.
- **Low bandwidth**: 6lowpan operates over the 2.4 GHz, 915 MHz or 868 MHz bands and it supports data rates 250 kbps, 40 kbps, or 20 kbps respectively.
- **Two available network topologies:** A 6lowpan supports both star and mesh network architecture.
- **Low cost:** The hardware cost is normally low. Hardware is compromised from a radio antenna, a sensor and so on.
- **High density support:** 6lowpan enhances addressing allocation and is able to operate efficiently in high density networks.
- **NAT:** Nat addressing is not necessary.

However, 6lowpan standard is not quite simple in order to be integrated over the IEEE 802.15.4 standard. Authors, Kushalnagar and Montenegro, (2004), point the key problems that makes 6lowpan standard difficult to be integrated with the IEEE 802.15.4 standard:

- **IP connectivity:** There are not currently implemented methods that allow IP-based traffic over an IEEE 802.15.4 network. Moreover, IPv6 requires 1280 octets as payload while 802.15.4 only supports 81 octets payload packets long.
- **Interoperability:** The majority of the routing schemas cannot be implemented over the 6lowpan. The current service discovery techniques are not yet supported from 6lowpan.
- **Security issues:** The 6lowpan does not provide sufficient security mechanism that enables data confidentiality and integrity of data.

2.3. Middleware

The authors Hadim and Mohamed (2006) refer that "a middleware layer is a novel approach to fully meeting the design and implementation challenges of WSN technologies". Middleware comes to integrate the gap between the sensor network and the application layer (Figure 6)

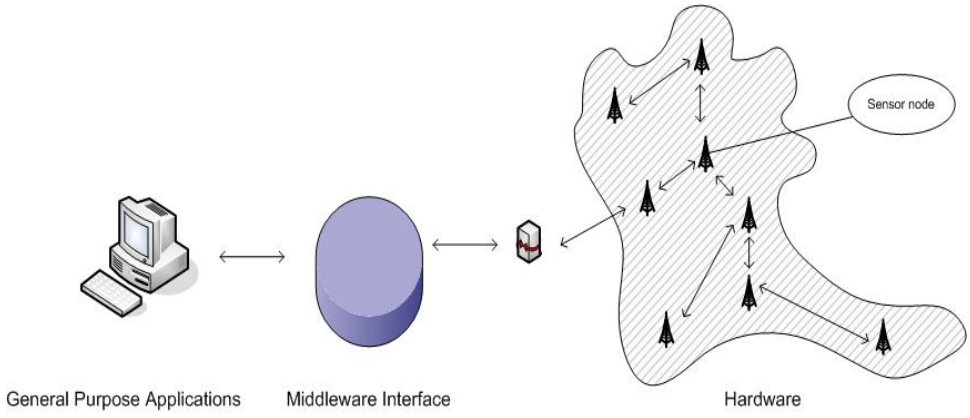


Figure 6: A middleware interface architecture

Today's computer-based devices both portal and non portal are well defined due to the fact that their functionalities operate over an operating system. An OS provides the availability to integrate the gap between the hardware components and the application layer. However, WSN available OS architectures do not provide the required interoperability between the hardware (the sensor nodes) and the application. A middleware architecture tries to cover seamless this gap. Such architectures are able to provide deployment, maintenance and execution of sensor based applications (Römer et al, 2006). Römer and his colleagues (2006) analyze the basic characteristic points that a middleware interface should have. First, a middleware interface should provide transparency among the sensor network and traditional networks. The majority of the traditional networks are IP – based. Akyldiz et al., (2002) agree that the interoperability of a WSN to other networks is in great research interest. However, current techniques such as the 6lowpan enables the interconnection with IP – based network. Römer et al., (2006) agree that 6lowpan requires complicated techniques and architectures in order to be able to provide a seamless connection to traditional networks. Instead, of applying complicated techniques for monitoring and deploying a 6lowpan network, it is worth maintain an intelligence middleware interface. A middleware interface will be based on preconfigured WSN architectures. Next, it should provide an easy access and control way of management and maintenance. WSN are based on a data – centric architecture schema. Sensor nodes task is to collect, aggregate and forward data to its neighbors. In this case a middleware interface is concerned as application-based. Further on, as a middleware interface acts as a management tool against a network that is probe to errors, it should yard with a self-configuration and fault tolerance mechanism. A number of WSN applications tasks require a real time data transfer. Middleware interfaces in such cases should implement time and location information control.

Current WSN`s applications are serve one task. This is also the case for the available middleware architectures. Yu et al (2006), refers that middleware interfaces should be multi – application. A middleware should be able to provide on the end user the support of more than one application with the minimum defects. The authors Hadim and Mohamed (2006) extends the basic characteristics for a middleware

interface in terms of openness and ease of use. The interface should be simple, avoiding low level complexity, providing an ease to use way for management and controlling. Functional requirements are closely related to the middleware interface. Openness should provide seamless integration of the middleware with new requirements. In balance a middleware interface should also adopt the following three functional characteristic keys. These are:

- **Power limitation**: A middleware interface should take under consideration the limited power capabilities of sensor networks. Power off nodes, nodes in sleep mode, new added nodes should be controlled and managed without interrupting the overall network operation.
- **Scalability**: The nodes number of a network may vary. A middleware interface has to manage and control efficiently any expansion of nodes. It also should reduce the complexity in case of high nodes density.
- **Topology self – configurations**: A middleware interface should implement a localization mechanism for the sensor nodes at the first nodes` deployment. Such mechanism should enforce normal and low power operation of the overall network.

Key Characteristics	Short Description
Interoperability	A middleware interface for WSN has to enforce interoperability between wireless sensor networks and traditional wireless networks. This interface will first implement existed WSN architectures and second will maintain an easy way for application controlling.
Node application – based	A WSN middleware interface should hold a most specific application – based schema.
Self – configuration and false – tolerance	WSN middleware should to provide self – configuration and false – tolerance automation mechanisms as nodes in a WSN are probe to dynamic changes due to power limitations, topology reallocation and others.
Integrate a time – location management	WSN applications, usually, focuses on real – time data transfer. In the most cases, time and location information are needed. Thus, a WSN middleware should integrate a time – location management

control	control.
Multi – application	A middleware should support user control for two or more applications concurrently allowing in such way to exploit at the maximum the capabilities of the WSN.
Ease of use	Easy to use refers to the middleware interface complexity. Seamless integration of the low – level APIs for heterogeneous networks. The interface in general should provide an easy way for network management and control.
Openness	Openness refers to the dynamic changes for a middleware interface when the functional requirements are changed. This changes should be seamless to the network.
Power limitation	A middleware should take under consideration any power limitations for the WSN.
Scalability	Candidate middleware interfaces have to manage and control the integration of a high dense WSN.
Topology self – configurations	A middleware should enforce localization techniques. Such techniques may provide an autonomous localization methodology for the sensor nodes after the first deployment.

Table 2: Key characteristics for a middleware

It is already a great research effort for managing and development middleware interfaces for WSNs. Maté (Levis and Culler, 2002) is a virtual machine (VM) that operates on the top of TinyOS. Maté is concerned as an event – driven interface, which allows users to build VMs. Introducing VM architecture, Maté reduces the code need to be applied on nodes while it provides safe execution (Figure 7). Code capsulation, is important for large programs, which data easily injected into the network. In addition, the code is split into capsules of 24 instructions. Each capsule length is one byte long. Maté is composed of two stacks: the operand and the return address stack. The majority of the instructions operate on the operand stack. In a Maté VM, capsules are forwarded themselves. A built – in routing algorithm take place while there are mechanisms for writing new instructions. Maté VM is either supported from Mica and rene2 platforms.

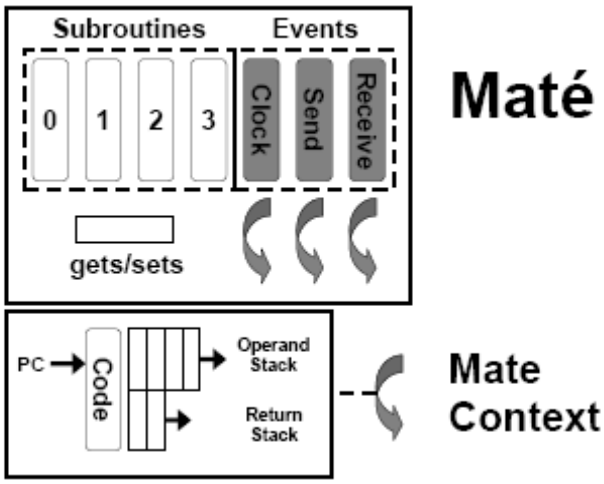


Figure 7: The Maté architecture (Levis and Culler, 2002)

Magnet (Hadim and Mohamed, 2006) is also a VM – based middleware interface. Magnet uses the Java VM and allows the network to appear as one. Static components are responsible for Java application-based development and objects formatting. After the objects creation, they are injected into the network. On the other hand, the dynamic components, are responsible to monitor and control each new object creation and invocation. They also provide the requested services for the application. Magnet, is implemented on the top of MagnetOS. MagnetOS is a power aware OS for WSN.

Cougar middleware implements a database – based approach. Cougar implements on each node a database. This approach results to a relational database of nodes. A querying mechanism is applied with a SQL – like syntax. Each node embedded database contains control and sensed data. Cougar interface relates data with the participated node which sense the environment plus any environmental characteristics. Cougar implements a power awareness mechanism. This is performed by multicast an in – network query. The multicast query reaches each node on the network. In turn each node collects data and forwards the result to a central node (Heinzelman et al, 2005). SINA, in contrast follows a hierarchical clustering approach, for data collection. Each cluster participated on the network, performs data aggregation. Aggregated data from each cluster is forward on a central node. This approach reduces data re – transmission of similar gathered data from the neighbor nodes (Heinzelman et al, 2005).

TinyDB (Maden et al, 2003) is a database – based middleware interface which implemented on the top of the TinyOS. TinyDB is Java – based interface, which integrates SQL – like queries to be applied directly on the network from the user. A key characteristic which is included in the TinyDB interface is an integration mechanism of a metadata catalog. This catalogue is responsible for holding all kind of sensors data. This allows an easy to use way for performing queries avoiding any in – network knowledge. TinyDB also implements a build in routing table on each node for its neighbors. This allows nodes to implement topology aware routing with minimum bandwidth utilization. In case that a node is not able to respond on a query directly, its neighbors inform it by forwarding a data packet copy and prompt it to start running.

```
SELECT tmp, AVG(light), AVG(volume)
FROM sensors
GROUP BY tmp
HAVING AVG(light) > l AND AVG(volume) > v
```

Figure 8: A short SQL - like script (for the TinyDB)

AutoSeC (Automatic Service Composition) (Han and Venkatasubramanian, 2006) middleware interface is database - QoS – based. AutoSeC basically inherits the characteristics of a traditional network. It enables application sharing, while maintains the QoS. AutoSeC control mechanisms ensure high QoS by absorbing dynamic network changes.

DSWare (Li et al, 2003), is similar to the AutoSeC interface. The main difference is that DSWare provides a best effort QoS using a cluster – based network architecture. On DSWare, cluster nodes are transparent to the middleware as each cluster is represented as a node for the network. This approach, enforce both low network traffic consumption, as queries from users are performed on the cluster and not on each node available on the network. DSWare, main advantage is that implements low power consumption, as not all nodes` cluster participates on user

queries. DSWare is better than SINA interface (which is s cluster – based interface), especially for real time applications.

Impala (Liu and Martonosi, 2003) is a middleware interface that enables both the applications` modularity and adaptability. Impala was implemented as part of the ZebraNet wireless sensor network. It was implemented based on mobile code techniques while it is enabled to adopt any code changes dynamically. This was achieved by forwarding the updated data packets from the nodes to the interface itself. Impala is concerned as a lightweight middleware interface that was able to both improve reliability and minimize power consumption.

MiLaN (Murphy and Heinzelman, 2003) (Middleware Linking Applications and Networks) middleware interface is characterized by the ability to adopt any network changes on the available components and manage efficiently the components that remain active. MiLaN improves in such way the overall application performance. Its unique characteristic, in contrast to the rest middleware interfaces, is that it is able to control efficiently the network and adopt any changes in relation to the application demands. It actually separates the application layer from the network layer, managing seperately components needed from the application. MiLaN implements a graph for the application. It also graphs the low level components, enabling the determination of the available utilities for these components.

Mires (Figure 9), is a message – oriented middleware (Souto et al, 2004). It implements a publish - subscribe communication model for the application. Mires, is able to encapsulate network protocols while provides a high – level API for development of applications. Publish and subscribe communication techniques are implemented using the data dissemination model. The sender node (the node who gathers the data) forwards the information to a predefined group of nodes on the network. In this case not all nodes get the message, instead only those nodes that already subscribe on the topic. Topics actually distinguish different kind of information that is available on the network. After this process completion, any that are subscribed on a topic are able to collect sensed data and forward it back to any node.

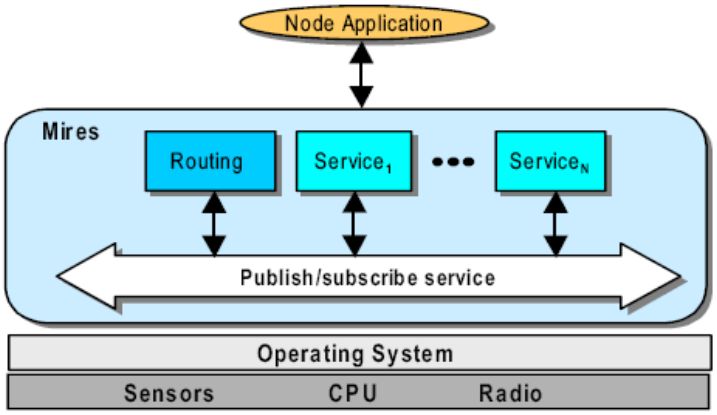


Figure 9: Mires architecture (Souto et al, 2004)

EnviroTrack (Hadim & Mohamed, 2006), is concerned as an object – based middleware interface due to the fact that it is implemented based on a programming abstraction. EnviroTrack includes a powerful programming development interface that allows the implementation of efficient tracking techniques for the environment. EnviroTrack implements an attribute – based technique for nodes naming where the addressing on network is not based on the nodes but on the data `s content. Using a deployed program, the user is able to control the monitoring environment, with the use of context labels on the physical targets. This technique allows EnviroTrack to be applicable on dynamic environments. EnviroTrack is built on the top of TinyOS implementing nesC-based programs.

Kairos is a middleware interface which implements a macro – programming model (Gummadi et al, 2005). It allows, in a centralized way both the control and management of the interface. Kairos, is easy to use environment, it hides complicated tasks, such as programming details for the distributed network code, data control and management and in network flow control. This centralized approach, allow Kairos to present each node on the network as an abstraction, this allows nodes to act concurrently within a single program.

OCTAVEX, is a service – oriented middleware interface. It allows any sensor node in the network to be added or removed from the existed network infrastructure. Additionally, it supports a collection of high level APIs for network programming, while it integrates preinstalled modules that allow an easy way of programming. An advantage of OCTAVEX is that it is able to manage and support concurrently

different types of protocols including both routing and network. Further on, it also implements a web – based interface, for remote management and monitoring..

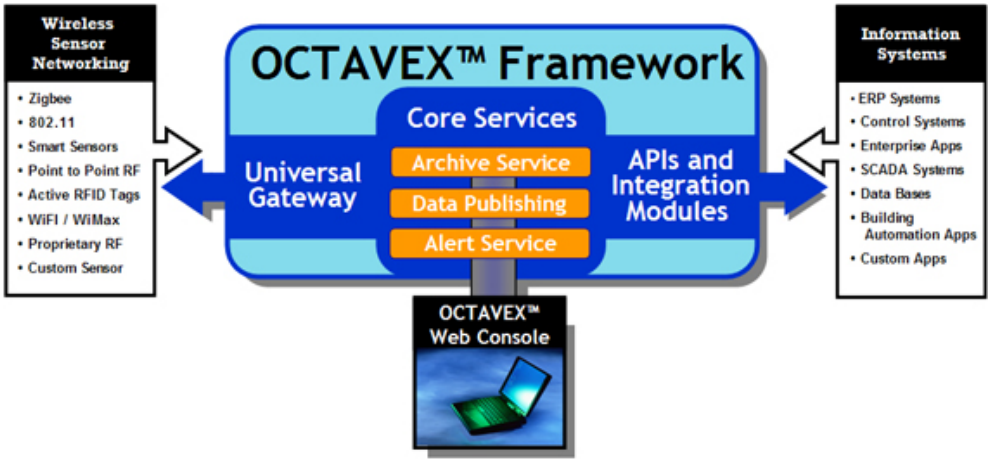


Figure 10: The OCTAVEX middleware architecture (OCTAVEX™ Technology, 2006)

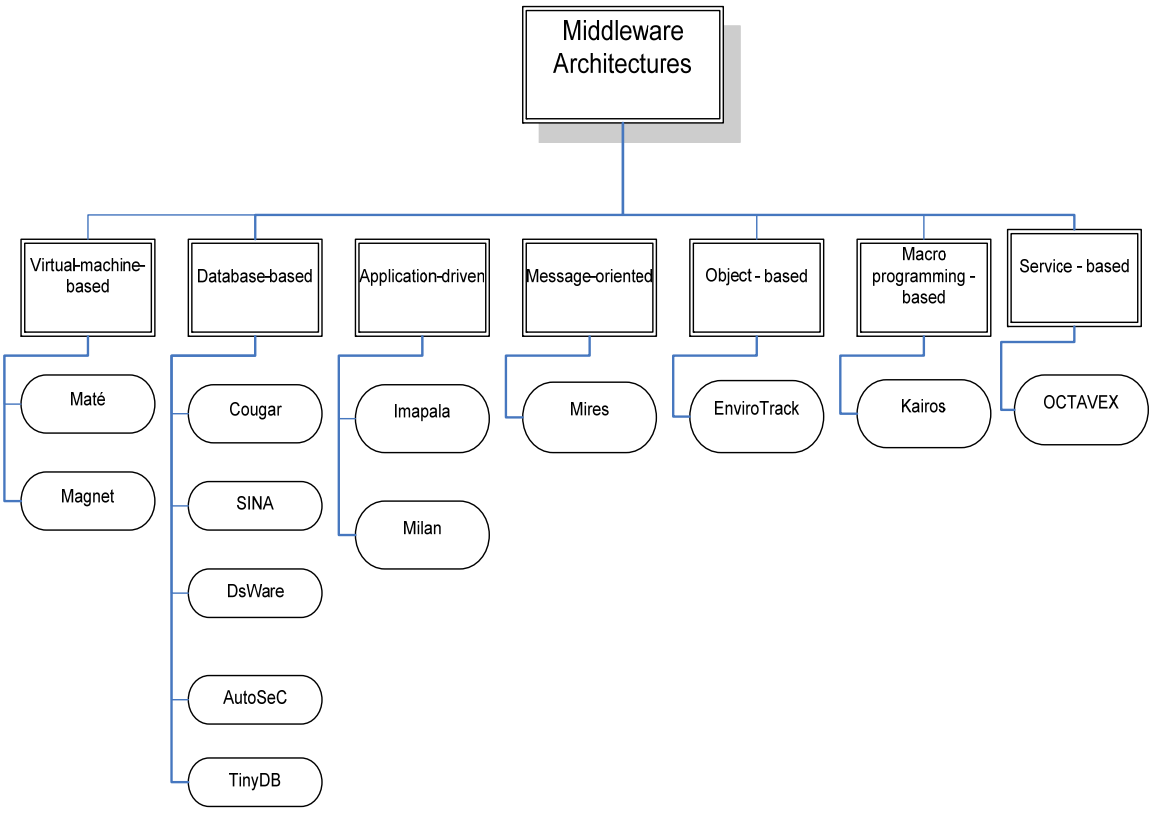


Figure 11: Available middleware architectures

2.4. Operating Systems

There is already developed a number of OS (Operating Systems) that allow the operation of applications on a WSN. Operating systems on traditional computers are primary try to interconnect user – based applications with the hardware. In addition current OS exploit at the maximum the hardware capabilities by providing seamless thread execution, powerful memory allocation and so on. In contrast operating systems available for WSNs are completely different from the previous ones. Culler (2006) highlights the basic characteristics that an operating system for a WSN should have:

- It should be able to manage and control a variety of applications,
- It should seamless allow heterogeneous platforms to efficiently operate.
- It has to provide easy to use and inerrable application development,
- It should follow and support the concurrency model,
- It should minimize the code size in order to improve performance.

TinyOS is a wide used operating system for WSNs (TinyOS, 2006). It is an open – source operating system developed from UC Berkeley Research Group. TinyOS inherits the component – based model, allowing code execution with low power loss and easy of management and controlling. Components such as protocols, hardware drivers, sensor board drivers and others are combined into libraries. TinyOS is an event – based operating system. It enables task scheduling and management. The TinyOS is completely implemented using nesC. nesC compact programming language is C - extension based language, applied for wireless embedded devices. The main design characteristics are listed below.

- TinyOS takes into consideration WSNs constraints including hardware constaints. Such constraints may be power limitation, narrow bandwidth allocation and low processing power.
- It enhances heterogeneity as far as it supports multiple hardware platforms.

- It also implements scheduling tasks that allow the error free communication among the nodes and the sink.

Beyong the TinyOS, there are a number of available operating systems. Some of them are Bertha (pushpin computing platform) Contiki, CORMOS, BTnut Nut/OS, eCos, EYESOS, MANTIS, SenOS, MagnetOS , SOS, t-Kernel and LiteOS.

2.5. Applications in WSN

Akyildiz et al (2002) recognizes three potential types of application protocols relatively to the following categories: a) The Sensor Management Protocol (SMP), b) The Task Assignment and Data Aggregation Protocol (TADAP) and c) The Sensor Query and Data Dissemination Protocol (SQDDP), which serve a portion of the tasks that are required in WSN.

The sensor management protocol defines the communication means between the administrator and the network. Via SMP the administrator can introduce new rules to the sensor nodes, perform time synchronization, and on / off switch the sensors, performing network configuration and managing security issues. On the other hand, the Task Assignment and Advertisement protocol is responsible for the way the data is been transmitted from the sink to the nodes. The TAAP is closely related to the lower layer operations such as routing. The Sensor Query and Data Dissemination Protocol (SQDDP) is the interface between the user and the whole network. Regarding to the ways described above (querying and tasking applications) the TAAP is responsible for managing queries from the user to the sensor nodes.

Authors, Mainwaring et al (2006) deploy 32 mica sensor nodes (University of Berkeley) on Great Duck Island (the project name was actually the Great Duck Island (GDI) system) for storm petrel monitoring. All nodes communicate with a

central node (gateway) named as CerfCube. The CerfCube was responsible to transmit data collected from the network to a central database using a satellite link.

Wang et al (2003) deployed preprocessing methods for the GDI system. They argue for a 2 – tier network in relation to the collaborated signal and information processing. They try to reduce the amount of data transmitted to the motes while they perform event filtering using the cross – zero rate, this method seems to be effective when high volumes of data are transmitted.

The University of Hawaii (Wang et al, 2003), research group deploy a WSN for spices plant monitoring. The research goal was to observe the fact that some plant spices grow up faster in comparison to the same species on a neighbor area. The research project was called PODS and it took place in Hawaii Volcanoes National Par. Personal Area Networks are used for communication, like Bluetooth and WLAN (802.11b). Based on a new designed protocol named as Multi Path On Demand Routing (MOR), the network was able to collect weather and image data and forward it to a remote database on the University of Hawaii. The research team also tries to investigate the relation between the sampling distance and the communication radius. According, to the project findings, the researchers show that the way that nodes are placed on field is chosen is related to distance and communication radius (Biagioni. and Sasaki, 2003).

All the above are all concerned as habitat monitoring applications. They are all based on a tiered architecture while they monitor dynamically changes in a phenomenon.

Environment Observation and Forecasting Systems (EOFS) are deployed in large geographic areas. Their goal is to monitor these areas relatively to physical processes such as environment pollution and so on. Common EOFS consist of 3 basic components: the sensor nodes which comprise the whole network, a distribution network and a central processing node (Xu, 2003).

CORIE project (Xu, 2003) is concerned as a prototype of EOFS. The project scope was to deploy thirteen nodes across the Columbia River. Sensor nodes are attached on a solar panel. The solar panel removes completely any power limitation issues for the network. Data collection was used for boats transportation mapping and weather forecasting. CORIE is concerned, without doubt, as a complex

application, especially in terms of management and control. The CORIE project is revised in a new version in order to adopt power supply issues for nodes and high topology demands.

The ALERT (Automated Local Evaluation in Real Time) (Roark, 2003) application has been developed by the National Weather Service. The goal of that project was to provide real time data transmission in concern to environmental issues such as rainfall and water level information. The sensor` s data transition is performed via an Infrared interface to a central sink. ALERT is concerned to be one of the major projects in WSN and it is extensionally used in USA usually for flood alarming.

Beyond habitat monitoring and environmental forecasting, health applications hold a great portion of interest from the medicine scientific field. Health oriented applications include doctors and patients monitoring in a hospital, monitoring of the disposal of drugs and patients monitoring relatively to pathological data collection.

The SSIM (Schwiebert et al, 2001) (Smart Sensors and Integrated Microsystems) project was a biomedical application, with the goal to monitor the artificial retina. A hundred of micro sensors were built and implemented within the human eye. Patients with vision problems were then allowed to see in an acceptable level. The modulation technique which was used in SSIM was the TDMA due to the fact of its efficient energy management of the motes.

Even if SSIM and an additional number of health projects are employed (including Glucose level monitors, Organ monitors, Cancer detectors and General health monitors (Xu, 2003)) for enhancing medical issues, there is a great number of issues involved before speaking for an effective health application. These include the safe and reliable communication, minimal maintenance and energy support from the human heat (Xu, 2003).

SHM (Rytter, 1993) (Structure Health Monitoring) system `s goal is to monitor damage, localize damage area, estimate the extent of the damage and evaluate the remaining life of the damaged body organ. SHM is a high cost system, approximately costs $ 25 million and it was first proposed in 1990.

In concern to home applications, (Mani et al, 2001) deployed a project under the name Smart Kindergarten. Smart Kindergarten focuses on the early childhood education.

2.6. Hardware Platforms

Industry of microelectronics has developed low cost devices called Multifunctional Sensor Nodes (MSN). MSN are able to sense their environment, collect and aggregate sensed data and forward it over a wireless communication protocol to its neighbor nodes and at the very end to the sink (a central node with higher computational and memory power than the other nodes on the network).

There is a great number of manufactures for MSN, offering a wide range of embedded microcontrollers. Even if, one of the key characteristics for developing a WSN is the price, the majority of the available hardware is still expensive. The up – to – date MSN available hardware platforms are the following.

RF mote: RF (Radio Frequency) (Hollar, 1996) motes, was the first available hardware platform generation for WSN development. RF motes are designed since 1999. They are compromised from an Atmel AT90LS8535 microcontroller, offer an RF 916 MHz transceiver and seven sensors including temperature, light, barometric pressure, 2 – axis acceleration and 2 – axis magnetometers. The lifetime of an RF mote was either five days for continuous operation or one and a half year at 1% duty cycling, using a 3V lithium coin cell battery. Motes from 5 to 30 meters distance are able to communicate with each other at a rate of 5kbps. RF motes are probe to collisions due to their communication pattern which was made upon the same carrier frequency, allowing in such way only one mote to communicate each time. The CRC (Cyclic Redundancy Check) algorithm is used for checking the packets integrity.

Miniature Motes: Miniature motes (Hollar, 1996) were an improvement of the RF node, designed both from S. Holler and C. Adela. Their key characteristic is that they are smaller and thus their price was lower. A miniature Mote, contain an RF transceiver, a microcontroller (Atmel AVR AT90S2313) and a temperature sensor. In addition, the communication bandwidth was 10 Kbps. The next generation of the miniature motes was the weC mote.

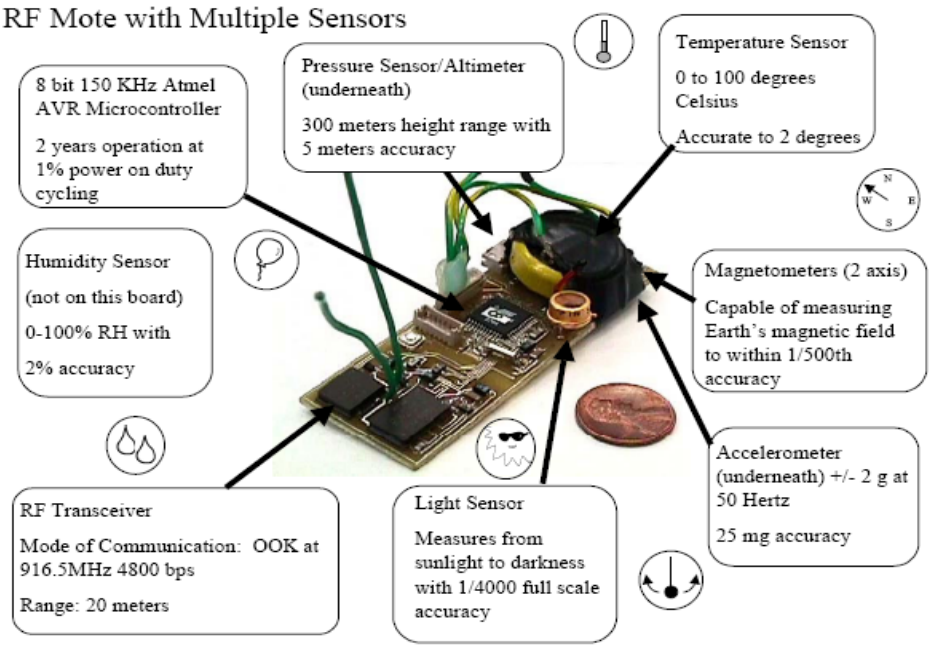

RF Mote with Multiple Sensors
8 bit 150 KHz Atmel AVR Microcontroller
2 years operation at 1% power on duty cycling
Pressure Sensor/Altimeter (underneath)
300 meters height range with 5 meters accuracy
Temperature Sensor
0 to 100 degrees Celsius
Accurate to 2 degrees
Humidity Sensor
(not on this board)
0-100% RH with
2% accuracy
Magnetometers (2 axis)
Capable of measuring Earth's magnetic field to within 1/500th accuracy
Accelerometer (underneath) +/- 2 g at 50 Hertz
25 mg accuracy
RF Transceiver
Mode of Communication: OOK at 916.5MHz 4800 bps
Range: 20 meters
Light Sensor
Measures from sunlight to darkness with 1/4000 full scale accuracy

Figure 12: RF Mote with Multiple Sensors (Hollar, 2000)

Figure 13: The miniature mote (Hollar, 1996)

Laser mote: Laser mote (Hollar, 1996), is a variation of miniature mote enabling long distance communication using a laser. A laser mote is an autonomous mote that contains four sensors (humidity, light, temperature and pressure) and uses a standard Atmel 90LS8535 microcontroller. Laser transceiver – receiver have to be manually configured for achieving direct communication. Researchers demonstrate a one – way communication at the range of 21 Km.

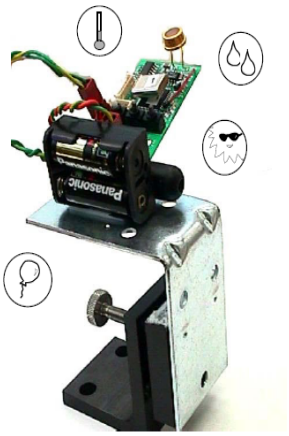


Figure 14: The laser mote (Holler, 1996)

CCR mote: CCR mote (Martin, 2004) enables passive laser communication using a CCR (Corner Cube Reflector). CCR motes only include a temperature sensor. The motes communicate with each other using laser beams. The transceiver sends a laser bean with the requested data, while the receiver reflects the beam to the transceiver with the required data.

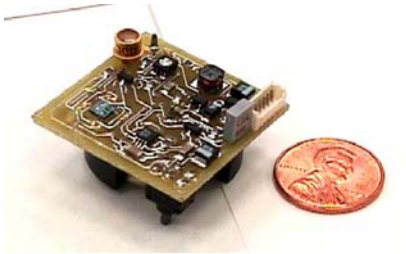


Figure 15: the CCR mote (Berkeley, 2006)

weC mote: weC (Holler, 1996) mote was bigger in size than the miniature mote. weC contains two sensors (temperature and light) and an PCB antenna for better communication between the nodes. Additionally, the use of a 4 MHz CPU clock, allow weC to be the most computational powerful mote. Moreover, the Atmel microcontroller allows programmer for remote access and reprogramming.

Figure 16: The weC mote (Hollar, 1996)

MICA mote: Mica (Crossbow Inc., 2006a) is the next generation of weC, developed from UC Berkeley `s research group. MICA contains an Atmega 128L low power microprocessor. It operates either on 916 MHz or 433 MHz RF, offering 40 Kbps data transfer rate. Motes are able to communicate on the maximum distance of 30 meters. The microprocessor CPU clock runs on 4 MHz, while there are 128 KB of flash memory, 4 KB of SRAM and 4 KB of EEPROM. MICA motes can either be accessed or programmed via the TinyOS. The available sensors, for a MICA mote, are the photo, microphone, sounder, magnetic and accelerator sensors. They are attached on sensor boards, which can connected onto a MICA mote via a 51 – connector (Figure 17 - 20).

Figure 17: The MICA mote (Crossbow Inc., 2006a)

Figure 18: Sensor board [MTS310CA] (for MICA motes) (Crossbow Inc., 2006a)

Data collection and aggregation is performed via the interface board. The interface board task is to collect data and pass it to the computer using a 25 – pin parallel cable.

Figure 19: Interface board (for MICA motes) (Crossbow Inc., 2006a)

MICA2 mote: Is the next generation of motes provided by the Crossbow (Crossbow Inc., 2006b). MICA2, first, differ from its predecessor on the available RF bands. A MICA2 mote operates on 816/ 916 MHz, 433 MHz or 315 MHz frequency bands. Actually, MICA2 contains the same microcontroller as MICA does but it offers 38.4 Kbaud data transfer rate and communication range up to 152 meters. MICA2 motes can be programmed via the TinyOS and are concerned as a better solution for commercial deployment.

Figure 20: The MICA2 mote (Crossbow Inc., 2006b)

MICAz mote: MICAz (Crossbow Inc., 2006c) operates on the 2.4 to 2.4835 GHz ISM band and is compliant to the IEEE 802.15.4 protocol. The features that make it different from both MICA and MICA2 motes is that is uses DSS (Direct Sequence Spread) spectrum, avoiding high rates of interference and providing data security. In addition, it supports 250 kbps data rates while its communication range for outdoor applications is up to 100 meters. Finally, it can be accessed and programmed via the TinyOS.

Figure 21: The MICAz mote (Crossbow Inc., 2006c)

MICA2DOT mote: MICA2DOT (Crossbow Inc., 2006d) is the last generation available from Crossbow. It is quite similar to MICA2 architecture, except for its quarter – size (25 mm) form factor and the reduction of input and output channels. In addition, it contains an on board temperature sensor and battery monitor. Moreover, the MICA2DOT is compatible with MICA2 motes.

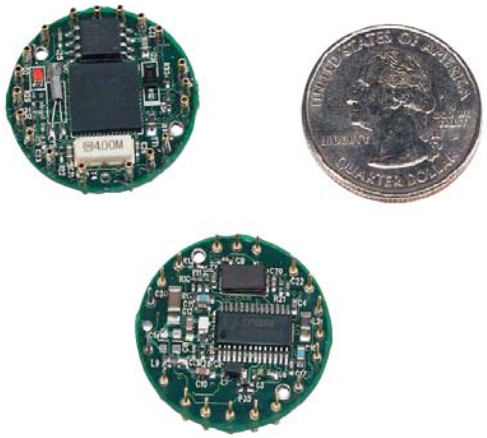


Figure 22: The MICA2DOT motes (Crossbow Inc., 2006d)

Spec mote: Spec mote (Yang, 2003) from UC Berkeley research group is actually based on a really small chip (approximately 2. 5 mm^2), named as the "smart dust" chip (Figure 23). This chip, despite its small size integrates sensors and transmitters. Spec is based on RISC architecture and offers 3 KB of memory. The communication band is on the 902 MHz. Spec can operate on a communication range of 12 meters, offering 19.2 Kbps data rate.

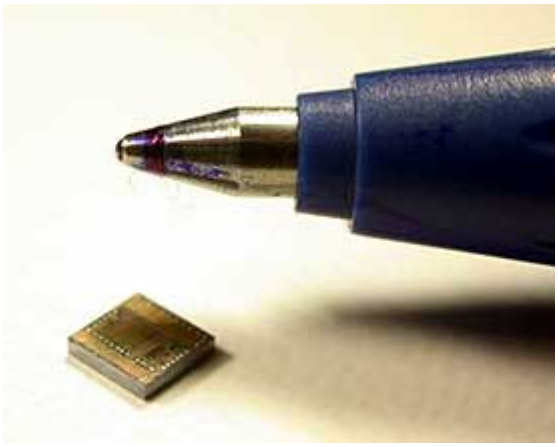


Figure 23: The "smart dust" chip (Yang, 2003)

Intel Mote: The Intel mote (Kling, 2006) designed and developed from UC Berkeley and the Intel Research Lab is a node platform that is targeted for a variety of applications. It uses the Bluetooth radio protocol (at the 2.4 GHz ISM band for WPAN) for communication, with a primarily focus of reducing power consumption. Intel mote contains a 32 – bit based ARM7TDMI microcontroller operating at 12

MHz. Thus, there is a 4x improvement of the performance in contrast to the MICA motes. In addition, it contains 64 KB of RAM and 512 KB of flash memory. The Bluetooth stack, allow high bit rates with maximum throughout about 720 kbps! Intel Mote software is based on the TinyOS and, in addition, it integrates a Bluetooth layer support.

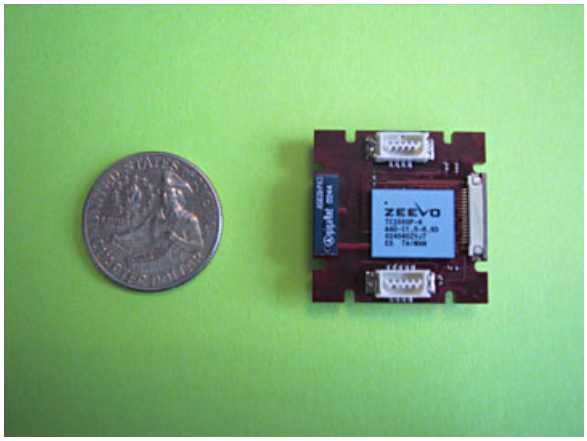


Figure 24: The Intel mote (Intel, 2006a)

Intel mote 2: Intel mote 2 (Intel, 2006b) is the next generation of high power motes developed from the Intel Corporation. Intel mote 2 contains a low power XScale processor the PXA27x on 13 MHz (for low voltage) and operates upon the IEEE 802.15.4 radio stack. The frequency can be scaled according to the supplied voltage, thus for lowest voltage supply the frequency starts from the 13 MHz and for the highest voltage supply increased up to the 416 MHz. It uses the 2.4 GHz ISM frequency band. It provides four available connectors on the mote board. On the one side, two “basic sensor board” connectors take place while on the other side are placed the other two “advance sensor board” connectors. Moreover, the processor includes a 256 KB of SRAM, divided into four equal parts of 64 KB, a max (up to this point) of 32 MB of SDRAM and 32 MB of flash memory.

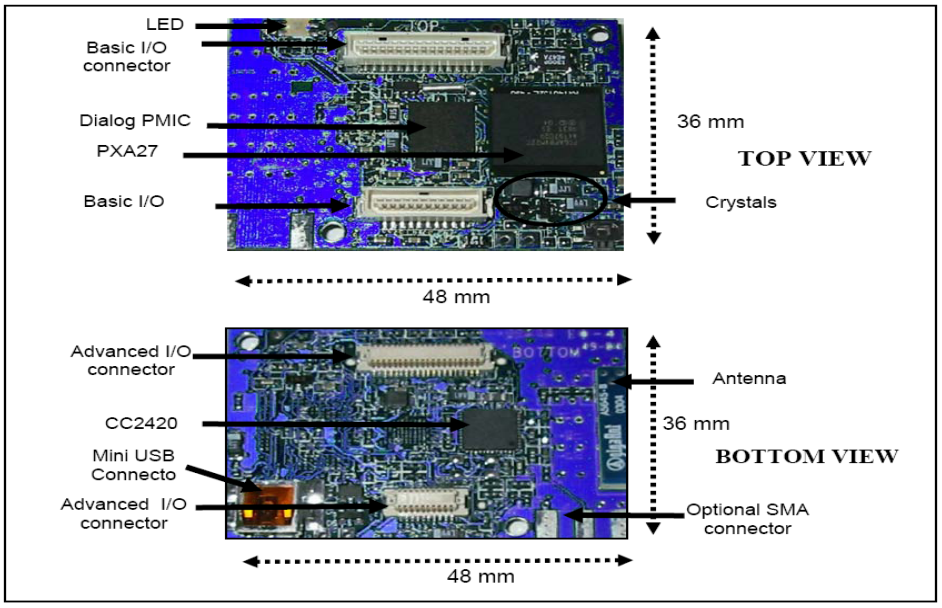


Figure 25: The Intel mote 2 main board (Intel, 2006b)

With no doubt, Intel Motes introduce a new challenge for the WSN. Integrating the Bluetooth technology makes it feasible to fully interconnect motes with current computing devices like PDAs and laptops. Moreover, the capability of high bit rates allow the introduction of WSN into real – time applications, providing the lowest latency. Intel, 2006c, provides a graph (Figure 26) which depicts a transfer rate comparison between Intel Motes and MICA family motes. The results, lead to a new, powerful and low consumption designing reality for MEMS.

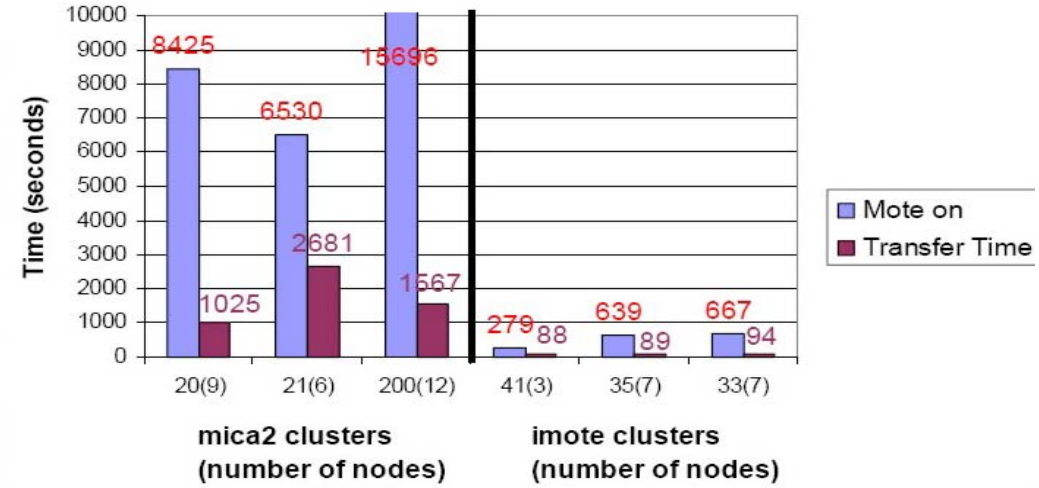


Figure 26: A comparison between the MICA and Intel Motes families (Intel, 2006c)

2.7. The x10 protocol

The x10 protocol allows the control for up to 256 devices. Each device is characterized from its area code (A - P) and device code (1-16). Management and control is performed by transmitting signals over a power line wiring. X10 protocol is first introduced in 1978 as part of the Sears control system (Chunduru et al, 2006). The x10 main advantages are that it is inexpensive and easy to deploy as communication among devices is performed upon the preinstalled electrical wiring lines. An x10 packet is composed of the destination address and an x10 command. Normally this command may be either “on” or “off” command. x10 data is encoded on 120 KHz carrier and transmitted as bursts.

An x10 system consists of a number of individual x10 devices. Data transmissions are synchronized on the AC power zero crossing point. Each node on the system uses a zero crossing detection mechanism that is used to listen on x10 – based data.

Bibliography

[1] 6LoWPAN, 2010. *6LoWPAN: IPv6 – based Low – power Wireless Personal Area Networks* [online]. Ajou University. Available from: http://6lowpan.net/ [Accessed 2 January 2010].

[2] ALFANDARI, L., PASCHOS, V., T., 1999. *Approximating minimum spanning tree of depth 2*. Intl. Trans. In Op.

[3] AL-KARAKI, J., KAMAL, A., 2004. Routing techniques in wireless sensor networks: a survey. Iowa State University. Hashemite University. *IEEE Wireless Communications*.

[4] ALTHAUS, E., FUNKE, S., HARPELED, S., 2005. *Approximating k-hop minimum-spanning trees*. Operations Research Letters 33.

[5] BAHL, V., 2006. *Philips; ZigBee* [online]. Philips. Available from: http://bwrc.eecs.berkeley.edu/Seminars/Bahl10.25.02/ZigBee.ppt#521,1, ZigBee [Accessed 10 May 2010].

[6] Berkeley. *Cots Dust, Large Scale Models for Smart Dust* [online]. Available from: http://www-bsac.eecs.berkeley.edu/archive/users/hollar-seth/ macromotes/macromotes.html. [Accessed 10 May 2010].

[7] BLUM, A., T., CAO, B., et al., 2005. *EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks*. Department of Computer Science, University of Virginia, Charlottesville.

[8] CHENG, H., LIU, Q., JIA, X., 2006. *Heuristic Algorithms for Real-time Data Aggregation in Wireless Sensor Networks*. ACM. IWCMC'06.

[9] Crossbow Technology, 2006a. *MICA* [online]. Available from: http://www.xbow.com/products/product_pdf_files/wireless_pdf/6020-0041-01_a_mica.pdf . [Accessed 15 May 2010].

[10] Crossbow Technology, 2006b. *MICA2* [online]. Available from: http://www.xbow.com/products/product_pdf_files/wireless_pdf/6020-0042-05_a_mica2.pdf. [Accessed 15 May 2010].

[11]Crossbow Technology, 2006c. *MICAz* [online]. Available from: http://www.xbow.com/products/product_pdf_files/wireless_pdf/6020-0060-02_a_micaz.pdf. [Accessed 15 May 2010].

[12]Crossbow Technology, 2006d. *MICA2DOT* [online]. Available from: http://www.xbow.com/products/product_pdf_files/wireless_pdf/6020-0043-04_c_mica2dot.pdf . [Accessed 15 May 2010].

[13]CULLER, D., E., 2006. *TinyOS: Operating System Design for Wireless Sensor Networks* [online]. Sensors. Available from: http://www.sensorsmag.com/sensors/article/articleDetail.jsp?id=324975&pageID=1. [Accessed 18 May 2010].

[14]GUMMADI, R., GNAWALI, O, GOVINDAN, R., 2005. *Macro-programmingWireless Sensor Networks using Kairos*. University of Southern California. Los Angeles.

[15]HADIM, S., MOHAMED, N., 2006. *Middleware Challenges and Approaches for Wireless Sensor Networks*. IEEE Distributed Systems.

[16]HAN, Q., VENKATASUBRAMANIAN, N., 2005. *AutoSeC: An Integrated Middleware Framework for Dynamic Service Brokering* [online]. Distributed Systems Middleware Group. Presentation. Available from: http://alamode. mines.edu/~qhan/research/talk/mdw01.ppt. [Accessed 2 May 2010].

[17]HEILE, B., 2005. *ZigBee Alliance Tutorial*. ZigBee Alliance.

[18]HEINZELMAN, W., B., MURPHY, A., L., CARVALHO, H., S., PERILLO, M., A., 2005. *Middleware to Support Sensor Network Applications* [online]. University of Rochester. Available from: http://www.futurehealth. rochester.edu/milan/IEEENetwork03.pdf. [Accessed 2 June 2010].

[19]HOLLAR, S., E., 1996. *COTS Dust* [online]. M. sc. Dissertation, University of California, Berkeley. Available from: http://www-bsac.eecs.berkeley.edu/archive/users/hollar-seth/publications/cotsdust.pdf [Accessed 10 May 2010].

[20]ILYAS, M., MAHGOUB, I., 2005. *Handbook of sensor networks: compact wireless and wired sensing systems*. CRC Press.

[21]Intel Research. 2006c. *Intel Motes and Wireless Sensor Networks* [Online]. Presentation. Available from: http://www.intel.com/research/downloads /snoverviewcd.pdf. [Accessed 15 February 2010].

[22]Intel, 2006a. *Intel® Mote* [online]. Available from: http://www.intel.com/research/exploratory/motes.htm. [Accessed 10 January 2010].

[23]Intel, 2006b. *Intel® Mote 2 Overview*, Version 1.0 [online]. Available from: http://www.intel.com/research/downloads/imote_overview.pdf. [Accessed 21 May 2010].

[24]KLING, R., M., 2006. *Intel Mote: An Enhanced Sensor Network Node*

[25]KORVIN, 2006. *Basic Lecture (ZigBee)* [online]. Available from: http://www.korwin.net/eng/infor/info_zb_03.asp. [Accessed 10 February 2010].

[26]KRISHNAMACHARI, B., ESTRIN, D., 2002. WICKER. *Modelling Data-Centric Routing in Wireless Sensor Networks*. IEEE INFOCOM.

[27]KUSHALNAGAR, N., MONTENEGRO, G., 2007. *6LoWPAN: Overview, Assumptions, Problem Statement and Goals* [online]. Network Working Group. Internet-Draft. Available from: http://www.ietf.org/internet-drafts/draft-ietf-6lowpan-problem-05.txt. [Accessed 10 February 2010].

[28]KUSHALNAGAR, N., MONTENEGRO, G., 2004. *6LoWPAN Overview, Assumptions, Problem Statement & Goals* [online]. Available from: http://www3.ietf.org/proceedings/05mar/slides/6lowpan-/6lowpan-4.ppt. [Accessed 10 February 2010].

[29]LEVIS, P., CULLER, D., 2003. *Maté: A Tiny Virtual Machine for Sensor Networks* [online]. University of California, Intel Corporation. Available

from: http://www.cs.berkeley.edu/~pal/pubs/mate.pdf. [Accessed 10 June 2010].

[30]LI, S., LIN, Y., SON, S., H. STANKOVIC, J., A., WEI, Y., 2003. *Event Detection Services Using Data Service Middleware in Distributed Sensor Networks* [online]. Department of Computer Science, University of Virginia, USA. Available from: http://www.cs.virginia.edu/papers/event_detect_dsn.pdf. [Accessed 11 June 2010].

[31]LIU, T., MARTONOSI, M., 2003. *Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems* [online]. Princeton University. Available from: http://www.cs.princeton.edu/~tliu/p71-tliu.pdf . [Accessed 10 June 2010].

[32]MADDEN, S., HELLERSTEIN, J., HONG, W., 2003. *TinyDB: In-Network Query Processing in TinyOS*. Version 0.4.

[33]MARTIN, G., 2004. *An Evaluation of Ad-hoc Routing Protocols for Wireless Sensor Networks* [online]. B.sc. Dissertation. Available from: http://research.cs.ncl.ac.uk/astra/Dissertation.pdf. Accessed 19 December 2006]. [Accessed 11 February 2010].

[34]MURPHY, A., HEINZELMAN, W., 2003. *Milan: Middleware Linking Applications and Networks*. University of Rochester.

[35]OCTAVEX™ Technology, 2006. *OCTAVEX™ Wireless Sensor Framework* [online]. Available from: http://www.octavetech.com/solutions/octavex.html . [Accessed 10 February 2010].

[36]PETROVIC, D., SHAH, R., C., RAMCHANDRAN, K., RABAEY, J., 2003. *Data Funneling: Routing with Aggregation and Compression for Wireless Sensor Networks*. IEEE.

[37]RAVI, R., MARATHE, M., V., RAVI, S., S., ROSENKRANTZ, D.,J., 1993. *Many birds with one stone: multi-objective approximation algorithms*. Proceedings of the 25th annual ACM symposium on Theory of computing.

[38]RÖMER, K., KASTEN, O., MATTERN, F., 2006. *Middleware Challenges for Wireless Sensor Networks*. Department of Computer Science. ETH Zurich. Switcherland.

[39]SOUTO, E., GUIMARΓES, G., VASCONCELOS, G., VIEIRA, M., ROSA, N., FERRAZ, C., 2004. *A Message-Oriented Middleware for Sensor Networks*. ACM. Toronto, Ontario, Canada.

[40]STOJMENOVIC, I., 2005. *Handbook of sensor networks; Algorithms and architectures*. John Wiley & Sons.

[41]TinyOS. 2006. *Mission Statement* [online]. Available from: http://www.tinyos.net/special/mission. [Accessed 25 May 2010].

[42]YANG. S., 2003. *Researchers create wireless sensor chip the size of glitter* [online]. Available from: http://www.berkeley.edu/news/media/releases/ 2003/06/04_sensor.shtml . [Accessed 10 February 2010].

[43]YEN, H., LIN, F., Y., LIN, S., 2005. *Efficient Data-Centric Routing in Wireless Sensor Networks*. IEEE.

[44]YU, Y., KRISHNAMACHARI, B., PRASANNA, V., K., 2006. *Issues in Designing Middleware for Wireless Sensor Networks* [online]. University of Southern California. Available from: http://www.cse.mrt.ac.lk/ lecnotes/cs5901/ readinglist/YuKrishnamachariPrasanna_middleware.pdf. [Accessed 10 February 2010].

[45]ZigBee Alliance, 2005. ZigBee V1.0 Architecture Overview. San Francisco.

[46]CHUNDURU, V., SUBRAMANIAN, N., 2006. *Effects of Power Lines on Performance of Home Control System*. IEEE Publications.

CHAPTER 3. METHODOLOGY

3.1. Introduction

The proposed system named as WiSecurity is based on a multi – application framework which acts as a middleware for a WSN. Actually, it incorporates the x10 standard with the sensor `s sensing capabilities providing a powerful autonomous security and fire protection system.

WiSecurity project, lays over a combination of operational layers for providing an intelligent way of exploiting at the maximum the readings coming from a WSN, in a real time way. It acts as a bridge between the user, the WSN and the x10 network at the same time, while it enables concurrently the management of both the sensor network and the x10 network over the Internet. It integrates a friendly - to - use environment while it completely covers the complexity of dissemination, collection and manipulation of data from the user.

WiSecurity enables the management of the WSN over the air. The user is able to adjust the system under his special requirements. Such requirements may include sensing interval window for deployed nodes and on node leds control. In addition, it enables the control of both the WSN and the x10 network over the Internet, using a symmetric encryption mechanism.

3.2. The Designing Phase

On this section we are going to define the designing phase of the WiSecurity System using UML. All the available activities of the system will be interpreted by two use case diagrams in relation to the available participated actors. Additionally, for each use case a high – level description will be presented on Appendix A.

On figures 27 and 28, we present the two main actors for the WiSecurity System. These are the local user and the remote user. The local user located on the server side (actually where the network is deployed). The remote user manages and controls the overall system over the Internet. For the majority of the activities, both the user from the server side and the user from the client side are able to participate.

The system actually is designed over three phases. The first phase was the WSN deployment and control, the second phase was the middleware interface integration and the last phase was the x10 network low – API integration. On the WSN deployment we use the TinyOS operating system. We program and upload a nesC – based code on each node. This application integration is responsible to manage node resources, manage dissemination of data and control incoming data packets. A central node, named as sink, is responsible to collect data from the network. The sink node is only responsible for data gathering from scattered nodes, flooding packets on the network and forward data packets over a serial port to a fixed interface (i.e. a computer machine). All nodes, except from the sink node, are power supplied. The end point for the WSN can be either a Linux or Windows based machine. The middleware interface actually listens on data communing from the attached serial port. Based on an easy to use interface user is able to manage and control the overall network either locally or remotely. Remotely communication is achieved over a secure communication channel. The middleware interface, integrates multiple applications on the same level. Such applications include the network management (i.e. nodes structure on the network), continuous data collection from the WSN (including temperature, noise detection, battery level, signal level and others), over the air data packets dissemination from the application to the network, nodes management based on TinyOS deploy – build – run Java – based frameworks, security and fire protection system management (including COMM ports, thresholds for data collection, email notification) and finally, an x10 – based manage application for allowing user to react directly with the x10 network. The third phase, the x10 network deployment, is composed of two end point devices and an x10 communication board. The communication board is attached on a local computer either via a serial or a USB port. Towards the two x10 - based endpoint devices we attach an alarm system and a sprinkler device. The Wisecurity, multiplication system is able to forward x10 messages to the x10 communication board via a Java – based interface. The x10 network is controlled either manually (user is able to manage x10 nodes – turn them on/off) or automatically via an embedded decision make algorithm. The algorithm`s operation can be parameterized by the user. The user is able to apply the security level for the WiSecurity system or manually assign

the required thresholds on collected data from the WSN. The decision – based algorithm just disseminates an x10 message to the x10 network in case, values from the WSN are upon the defined threshold. The x10 message could either turn on the alarm system or the sprinkler device. For example, in case of a sound detection event the alarm system is activated. Moreover, in case of sudden increase of temperature the sprinkler device is activated.

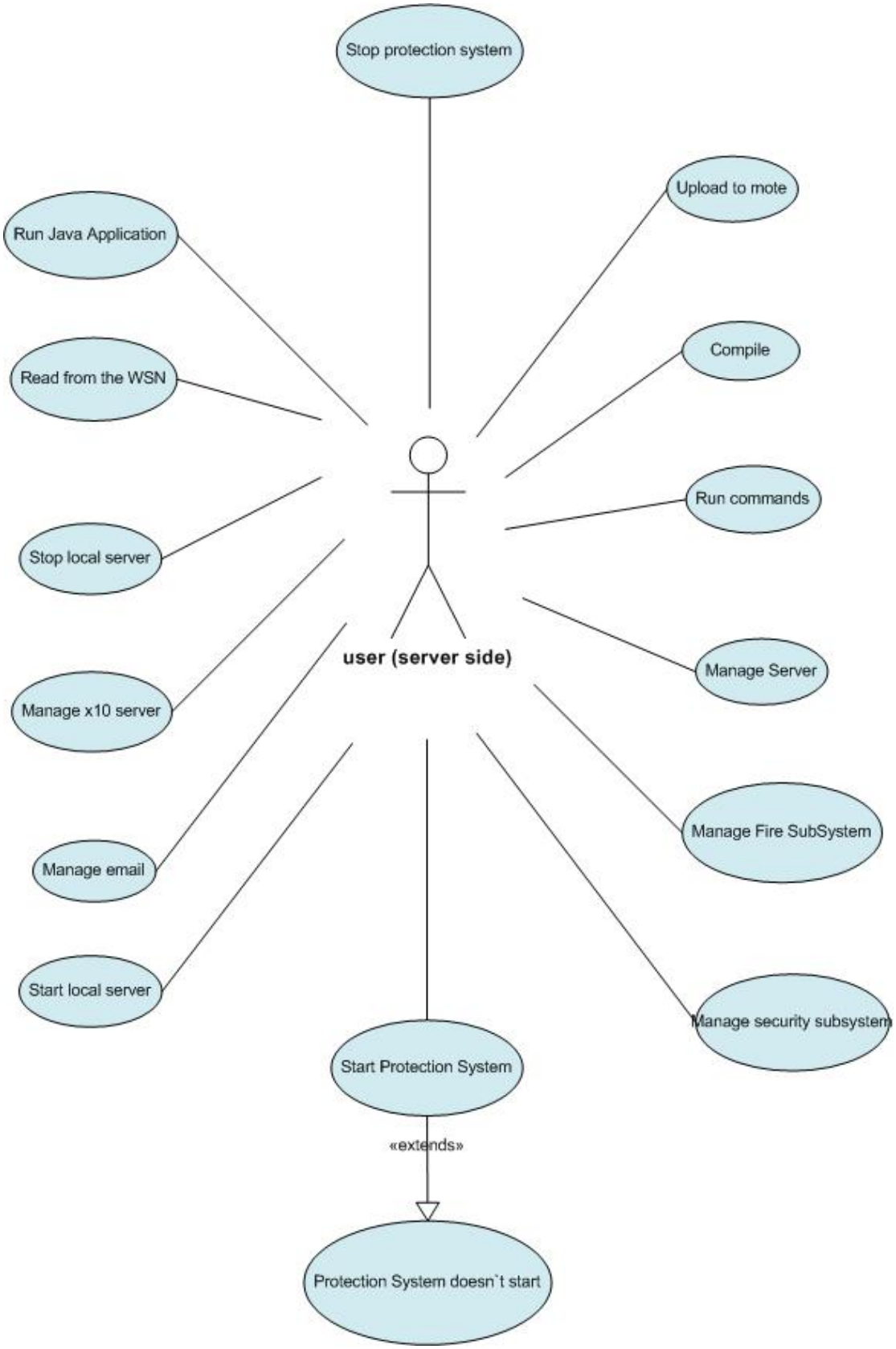


Figure 27: Use case diagram for the WiSecurity System I

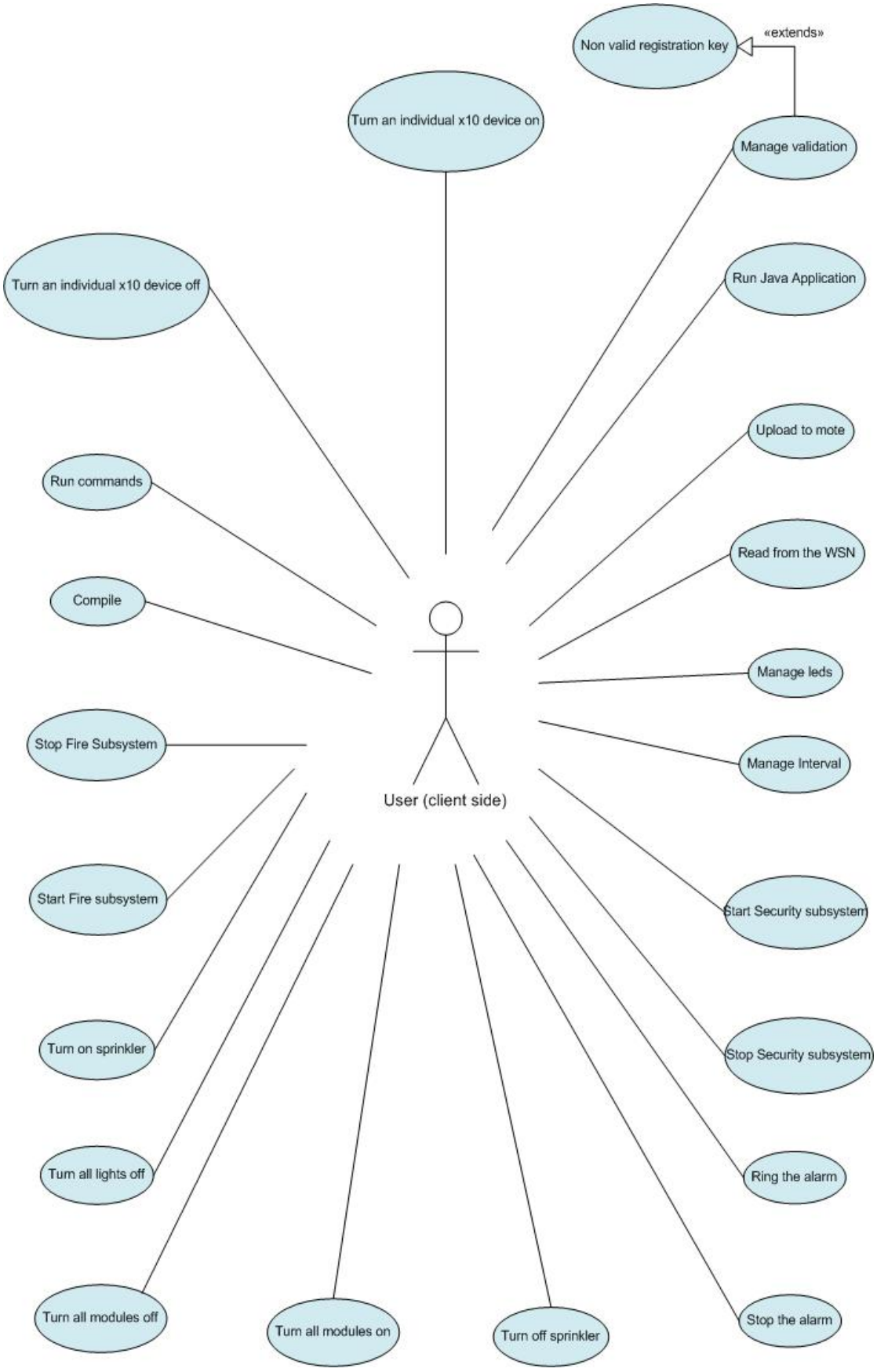


Figure 28: Use case diagram for the WiSecurity System II

3.2. The WiSecurity Layers

WiSecurity system integrates three layers. These are the WSN layer, the x10 layer and the Internet (TCP/ IP) layer (figure 29).

The WSN layer uses a collection of protocols for collecting/ disseminating data to/ from the root node of the network. It extends in such way the overall geographical coverage of the network. The root node acts as a bridge between the WiSecurity system and the WSN. Moreover, the route node is able to disseminate interest data to the WSN. Such data may include AM (Active Message) packets that control the led's status (turning them on or off) and Time Control Packets that are able to manage the sensing interval time window for nodes. Each node on the network disseminates data to the root node. Such data, involve AM packets regarding the temperature, the light intensity, the voltage, the microphone volume and the RSSI (Crossbow Inc., 2006b). Both temperature and light sensors; returns a real time value using the analog-digital converter channel 1 (ADC1) (Crossbow Inc., 2006a). Voltage readings come from the pin A5. The microphone sensor uses the ADC2 converter for disseminating units. RSSI; measures the signal strength between the nodes. The value for the RSSI is disseminated from the ADC0 converter.

WiSecurity system uses a number of formulas for converting the engineering units of temperature, light intensity, voltage and RSSI. Temperature value is converted into Celsius, light units into mVolt and voltage units into Volt. RSSI units are converted into dBm, for the measure of the power level and into Watt for the measure of the power consumption (Appendix B). The x10 layer is responsible for managing the x10 – based devices on the x10 network. Devices are able to be controlled in an autonomous way by the system. Moreover, system allows user to manage entirely the x10 network either locally or remotely, using a friendly to use interface. Remote communication is performed over the TCP /IP. Integrating the x10 protocol, WiSecurity is able to act on certain events coming from the WSN. A user, using the systems` interface is able to define the x10 address of the alarm device and the x10 address of the sprinkler device using a friendly to use interface. Such configurations are part of the security and fire protection subsystems. WiSecurity, system automatically turns on the selected devices when an event occurs. Such

events include the temperature changes on the deployed environment, illumination fluctuations and sound detection.

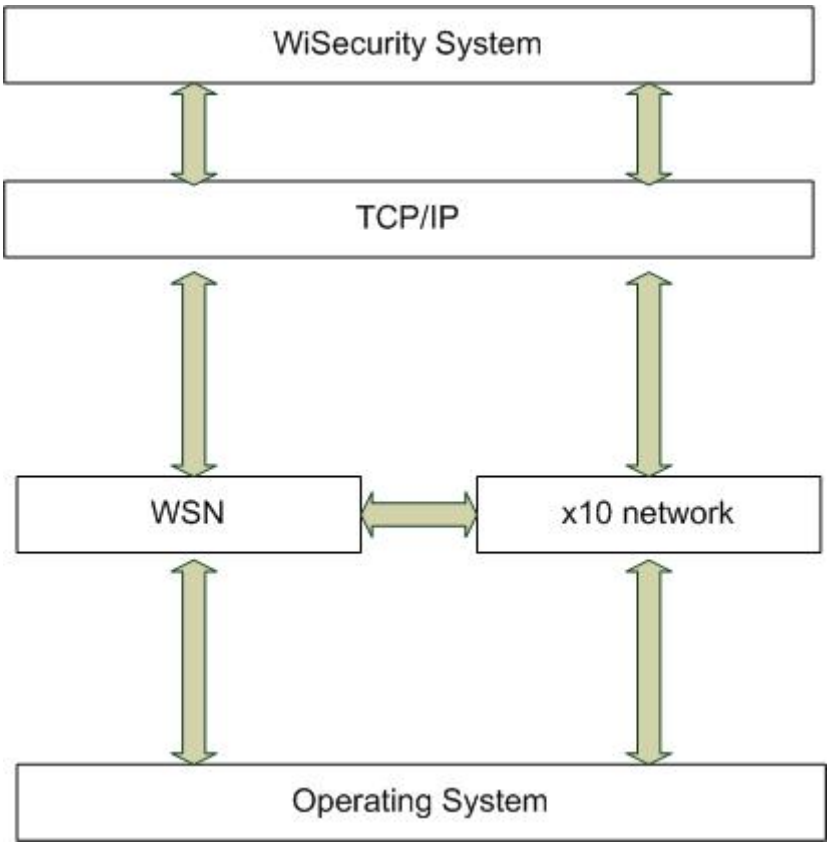


Figure 29: WiSecurity layers structure

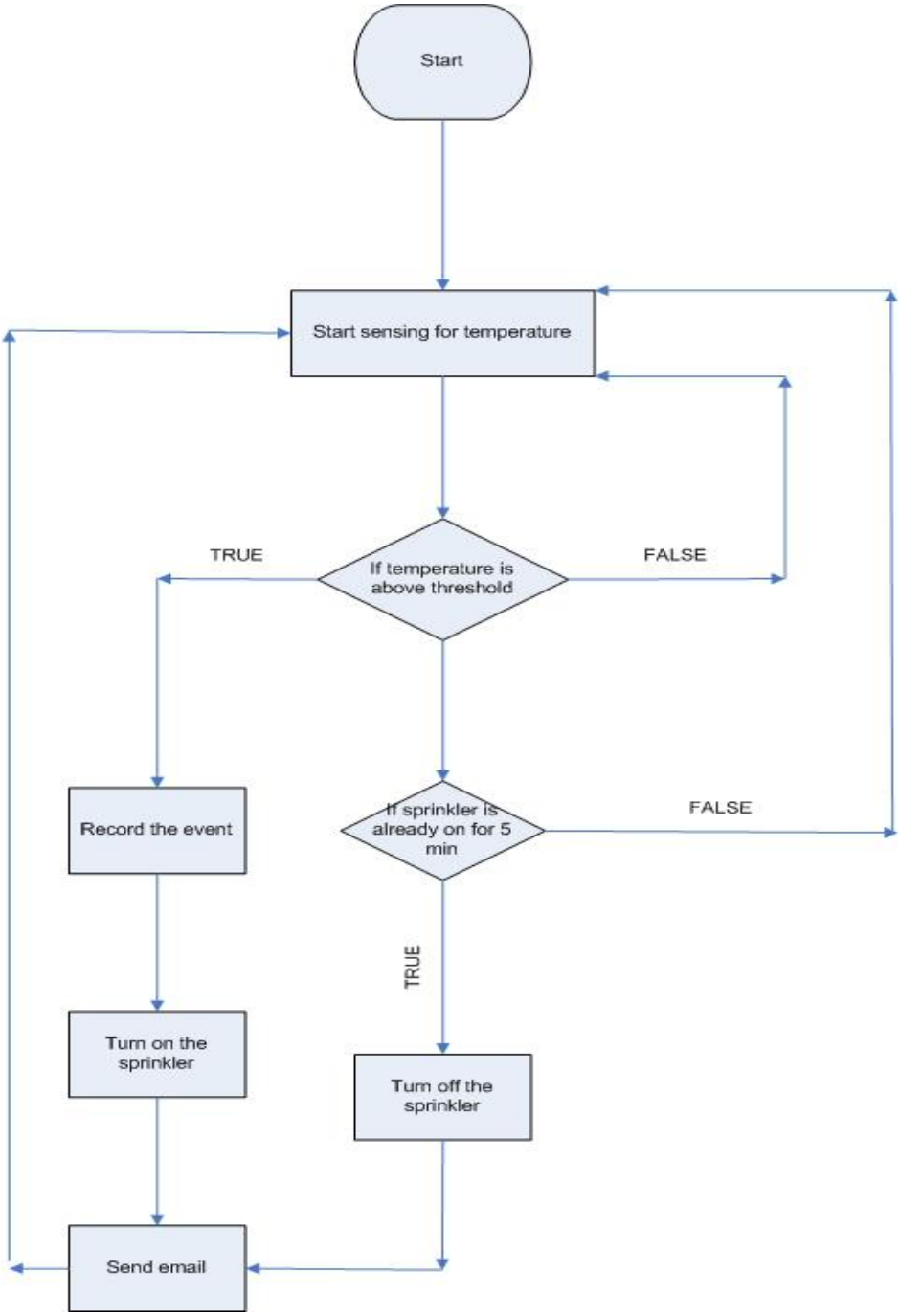


Figure 30: Flow chart for fire subsystem

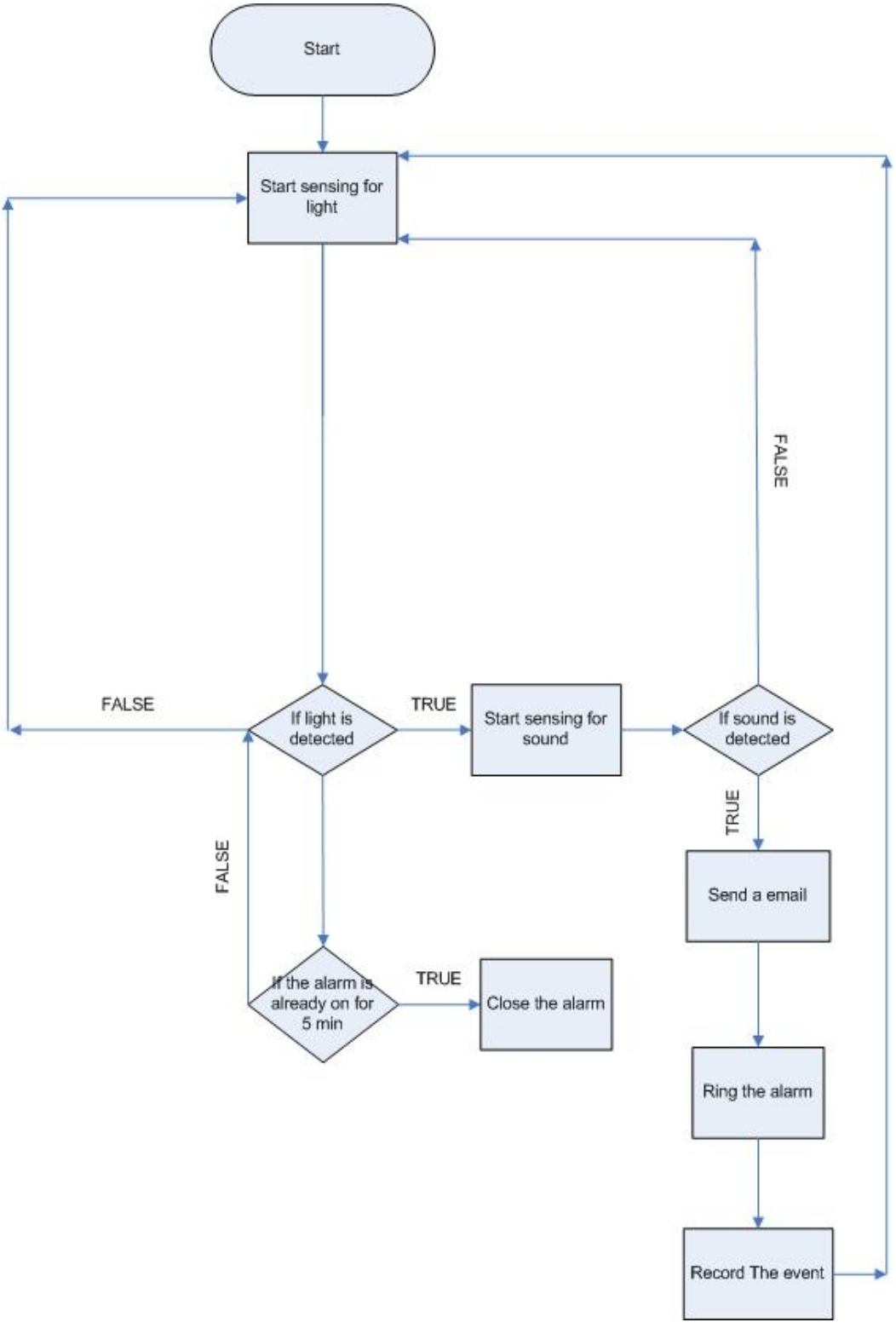


Figure 31: Flow chart for the security subsystem

3.3. The WiSecurity Subsystems

The WiSecurity System integrates four individual subsystems (Figure 32). These subsystems are:

a. **The x10 control subsystem:** This subsystem enables the local and remote control for up to 256 x10 based devices concurrently. In addition, this subsystem is used from the security and the fire protection subsystem auto decision algorithm.

b. **The WSN subsystem:** This subsystem is responsible for the control and management of the WSN either locally or remotely. Data is properly aggregated from this subsystem and can be exported on both the security and fire protection subsystems.

c. **The security protection Subsystem:** This subsystem is activated when one or more events take place on the WSN subsystem. In a real time way, the system compares the incoming light intensity value coming from the WSN subsystem, to an already stored maximum available limit. If the system, detects that the aggregated value for the light intensity is greater to the stored maximum limit, it starts the sound detection using the microphone sensor values. There are already predefined thresholds regarding the permitted values for a microphone sensor. The one is the maximum limit and the other the minimum limit for sound detection. If the aggregated data, is greater to the maximum limit and less to the minimum limit the alarm device is activated and a countdown chronometer starts. On this state, the system starts again to compare the light intensity value to the maximum limit. Finally, the system, continuously checks if the end time of the countdown is reached. If this is true, the alarm is closed (Figure 31).

d. **The fire protection Subsystem:** This subsystem aggregates the received temperature value from the WSN subsystem; if this value is greater to an already stored value then the sprinkler is activated. The stored value, depicts the maximum limit for temperature. When a sprinkler device is activated, fire protection system initializes a countdown chronometer. When the

chronometer reaches its end time, the system turns off the sprinkler device (Figure 31).

WSN, security and fire protection subsystems run in an infinite loop. Each node on the network is able to trigger the security protection subsystem or the fire protection subsystem or both of them when an event occurs on the WSN subsystem.

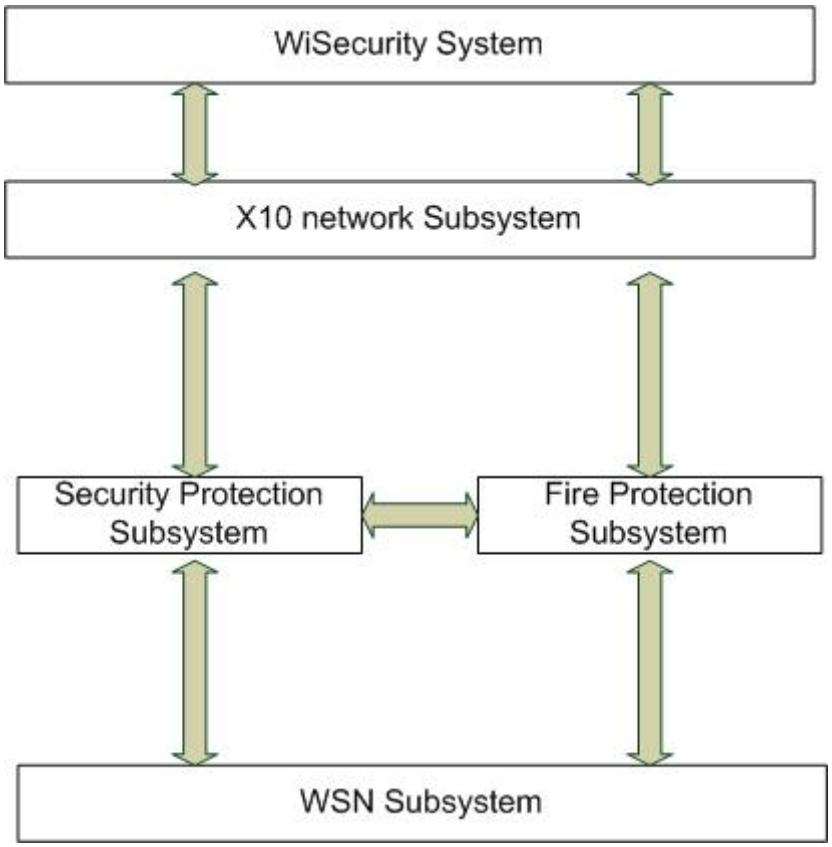


Figure 32: WiSecurity subsystems

3.3.1. The WSN subsystem

The WSN subsystem is responsible for the management and control of the WSN. It actually, implements a framework structure which is responsible for the control of the communication between the nodes and the base station. In addition, it manages the dissemination of data packets on the network and the collection of AM (Active Message) packets which are transmitted from the nodes in the WSN.

One or more nodes can be set us the root node (parent node) on the network. As a result, the rest of the nodes (child nodes) transfer the collected data, from their environment, to the root node closest to them. This is basically done, for the geographical expansion of the network. Implementing this collection schema, not all nodes have to be close to the base station.

Parent and child nodes are forced to sense their environment into user specified intervals, collect data, encapsulate it into an AM packet and transfer it to the destination.

An AM packet holds data regarding, the temperature, the light intensity, the microphone volume, the signal strength, the interval value and the version value. Values for temperature, light, microphone and signal strength are all available from the ADCs of the Mica2 or MTS300 board. Interval value, defines the interval between the sensing periods. Interval value is assigned on a default value when the WSN subsystem starts but at any time it can be changed from outside the network. In addition, the leds status can be changed from outside the network. Their status could be either set to on or off. Either interval value and leds status can be change, using a temporary value which included on any transferred AM packet from the base station to the WSN. Version value is a flag value used for changing the interval or leds status over the air. For example, when a root node receives a version with the value 0, it waits for the new interval, when the new interval received it disseminates the new interval to the whole network. After the end of the dissemination process, each node (child and parent) update their local interval with the new interval. This is actually, the dissemination schema which is used for the WSN subsystem. The dissemination schema is the reverse of the collection schema (Figure 33).

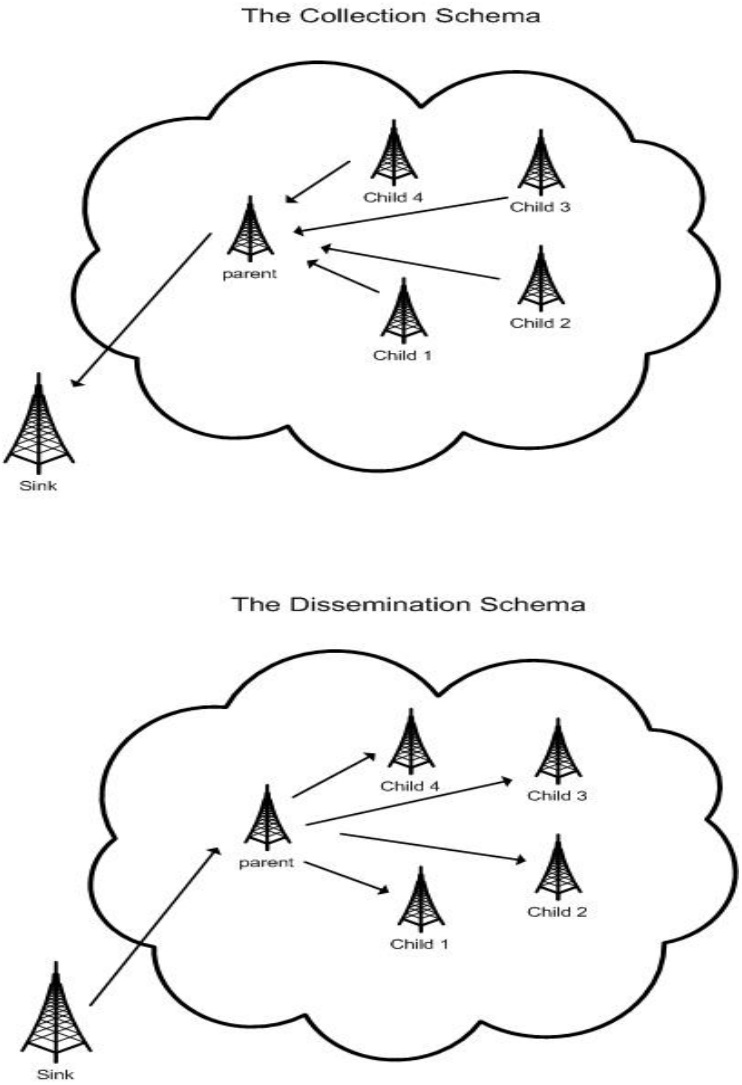


Figure 33: The dissemination and collection schemas

3.3.2. The x10 control subsystem

The x10 control subsystem manages the x10 – based devices on the x10 network. The security and fire protection subsystems are directly connected with this subsystem. It manages and control commands such as ALL LIGHTS ON, ALL LIGHTS OFF, ALL MODULES OFF or commands like A5 ON, where A is the address and 5 the code number of the x10 device for which the command is for.

When the security or the fire protection subsystem initialize one of the above commands, the x10 subsystem forwards it to the PC - to - x10 interface module. In turn, the interface module, transfer the command to the selected devices using the

x10 protocol structure (The x10 protocol communication schema is not part of this document, thus it is not discussed further).

The x10 subsystem can also be accessed remotely. WiSecurity system, includes a fully x10 – based control schema, enabling the management of the x10 network even if a user is out of the local network.

To summarize, the x10 control subsystem provides two functional requirements:

a. The control of all the x10 – based devices,
b. An autonomous way, of controlling one or more x10 – based devices from the security and the fire protection subsystem.

With the integration, of the x10 network, we exploit at the maximum, in a real time way, the collected data from the WSN subsystem, enabling in such way an autonomous decision system where acts as middleware between the WSN and the x10 subsystem.

3.3.3. The security subsystem

The security subsystem holds an important position on the WiSecurity system. It, continuously, listens to data regarding the light intensity and sound volume from the WSN subsystem. It uses a decision make algorithm, for evaluating the incoming data against any defined threshold. If the evaluation fails it calls the x10 control subsystem and in turn activates the alarm device.

A user is able, to specify the maximum and the minimum permitted values for the security subsystem. The system implements fixed size infinite loops for the light intensity data arrives from the WSN subsystem. At the end of each loop it makes an average of the aggregated data. If and only if this average is out of any predefined limit values, which already specified from the user, the system begins, again, in fixed infinite loops to collect and aggregate data coming from the microphone sensor. On the other hand if the average is inside the limits the system, continuous to aggregate the data coming from the light sensor. User is able to define the maximum and minimum permitted limits in case of sound detection. If the

system senses light and additionally detects sound, it turns for a fixed time of five minutes the alarm device on. The alarm device is managed from the x10 control subsystem. Additionally, the user is able to define up to 255 additional devices, which can be activated as soon as the alarm device is powered on and deactivated when the alarm device is off.
Finally, the WiSecurity system informs the user, automatically, using email notification messages for any event.

3.3.4. The fire protection subsystem

This subsystem follows, in some extend, the same idea as the security protection system. Using an infinite loop, it aggregates fixed size of continuous messages the temperature data coming from each one of the sensor nodes. The user is able to specify the maximum permitted temperature for the system. If and only if the average of the aggregated data from the fire protection subsystem is greater than the maximum defined temperature threshold, it initializes the x10 control subsystem. The x10 control subsystem, in turn, enables the sprinkler device for a fixed time interval. After the end of this time interval the x10 control subsystem, again, disables the sprinkler device. In case of high temperature detection, an email notification is sent automatically from the WiSecurity system.

3.4. The WiSecurity System

The WiSecurity system, actually integrates a WSN and an x10 – based network, which exposed as a middleware interface on the end user. It implements an intelligent automate decision making algorithm for potential threats, including protection from a burglary and fire. The system can be adjusted under any special requirements of the location where it is deployed, making it open on almost any requirement. According to the literature a WSNs middleware should follow some basic characteristics. On Table 35; we evaluate the WiSecurity system against those basic characteristics. We use a rate range from zero to five, for depicting minimum

or maximum fulfill against the middleware`s requirements respectively. The requirements for a middleware are discussed on section 2.2.

Key Characteristics	Short Description	Ranking for WiSecurity System	Notes
Interoperability	A middleware for WSN has to provide interoperability between sensor networks and traditional networks. This interface will first based on existed WSN architectures and second will provide an easy application – based way for controlling.	5	WSN subsystem, implements platform - independent nesC code, meaning that it can support any available platform. In addition, the management and control are both easy.
Node application – based	A WSN middleware should to integrate a most specific application – based schema.	4	WiSecurity acts as a data collector.
Self – configuration and false – tolerance	WSN middleware should to provide self – configuration and false – tolerance automation as nodes in a WSN are probe to dynamic changes due to power elimination, topology changes and so on.	2	WiSecurity informs the user for any potential errors such as out – of battery, but it is not dynamically adjusted to that changes.
Integrate a time – location management control	WSN applications, usually, focuses on real – time data transfer. In the most cases, time and location information are needed. Thus, a WSN middleware should integrate a time – location management control.	3	WiSecurity provides only time information.

Multi – application	A middleware could provide the user with two or more applications, concurrently allowing in such way to exploit the capabilities of the WSN.	5	WiSecurity enables the collection and management of the WSN, the control of an x10 network and an autonomous security and fire protection system.
Ease of use	Easy to use refers to the interface complexity, low – level APIs of heterogeneous networks. The interface should provide an easy to use way, providing a user friendly interface for managing and controlling.	5	A user is not aware of heterogeneities among the different network structures which are used. WiSecurity hides the complexity of subsystems integration from the application control.
Openness	Openness refers to the dynamic changes for a middleware when the functional requirements are changed.	5	WiSecurity can be adjusted on almost any functional requirement.
Power limitation	A middleware should take under account the power limitations of a sensor network.	1	WiSecurity assumes that the WSN subsystem is always power supplied.
Scalability	Candidate middleware interfaces have to analyze and calculate the maximum possible integration between a great number of nodes.	4	WiSecurity is able to integrate a great number of nodes. This can be done due to the fact that its node in the WSN, acts as an individual WSN for the system.
Topology self –	A middleware should provide	0	WiSecurity does not

configurations	an autonomous localization methodology of the sensor nodes after the first deployment has occurred.		manage or take under account any localization changes.

Table 3: Key characteristics for the WiSecurity System

The WiSecurity system is based on a client – server architecture. The communication between the two tiers is encrypted. For this an SSL Socket – based encryption is used, which is based on a 1024 bit key.

The server side does not act only as a forwarder of responses to the client, but it includes a complete collection of activities for managing the WSN, the security, fire protection system and the x10 network. A user is able to change the preferences of the system, according to his special needs. A user is able to define the WiSecurity server preferences, the x10 network preferences such as to assign the USB – to – x10 interface module` s port, the alarm`s and sprinkler`s addresses and so on. In addition, he is able to integrate different maximum or minimum limits regarding the light intensity, the sound detection and the temperature or to choose among the already default security levels. In addition, he is able to assign one or two email addresses; the system automatically will send email notifications when the alarm or sprinkler devices are activated or deactivated. Beyond, the available options, it is feasible to read, in real time, each data packet coming from the WSN on screen. Additionally, the server side integrates algorithms for the security, fire protection and x10 control subsystems. Both security and fire protection subsystems can be activated or deactivated at any time. A series of icons on the screen, inform the user for the enabled or disabled subsystems. Furthermore, a user is able to watch the connected clients with the server at any time, including the IP address of the client and the date/ time of connection. Next, an ease of read logs history holds information and error messages for all the subsystems. Finally, the system includes a series of tools for managing nesC – based applications and more. These tools include the capability of compilation of nesC code, the functionality of uploading application to a mote, running real time Java – based applications and running cygwin or TinyOS or nesC – based commands. All of these tools provide a friendly to use GUI, enhancement in

such way the capability of managing efficiently nesC code, making a step over the monolithic screen of cygwin.

The client side inherits some of the functionalities of the server side. In particular, it provides a more comprehensive way of management of all the available WiSecurity subsystems.

Initially, the client side acts as a remote middleware for both the WSN and the x10 network. Client is able to read data using a TCP/ IP network far away from the WSN deployed network. In addition, client is able to apply commands to the WSN remotely. Such commands include change of leds status (turning on / off) and the adjustment of the interval time between sensing periods. These commands are forwarded to the server side, and in turn, the server forwards them to the WSN subsystem. A series of icons, inform the user at any time about the overall system and the subsystems status. Additionally, the client is able to deactivate/ activate at any time one or more of the subsystems remotely, to ring the alarm or open the sprinkler manually. Beyond, the remote management and control of the WSN, the security and alarm subsystems from the client, a complete x10 control based consol is added. A client is able to manage all the light and appliance modules, remotely at any time. Available commands that are included are the ALL LIGHTS ON, ALL LIGHTS OFF, ALL MODULES OFF. Additionally, a client is able to manage up to 256 devices, concurrently, using a collection of toggle buttons for each of the x10 device separately.

With the WiSecurity system, three completely different networks (including WSN, TCP/ IP and x10) are integrated in a seamless way under the same system (Figure 34). It provides an easy to use environment; with a number of icons on screen which are used in order to inform the user about the status of the system (Figure 35). WiSecurity enables under the same application the management of the WSN, the collection of data from the WSN, the management of the security and fire protection systems and finally the management of the x10 network.

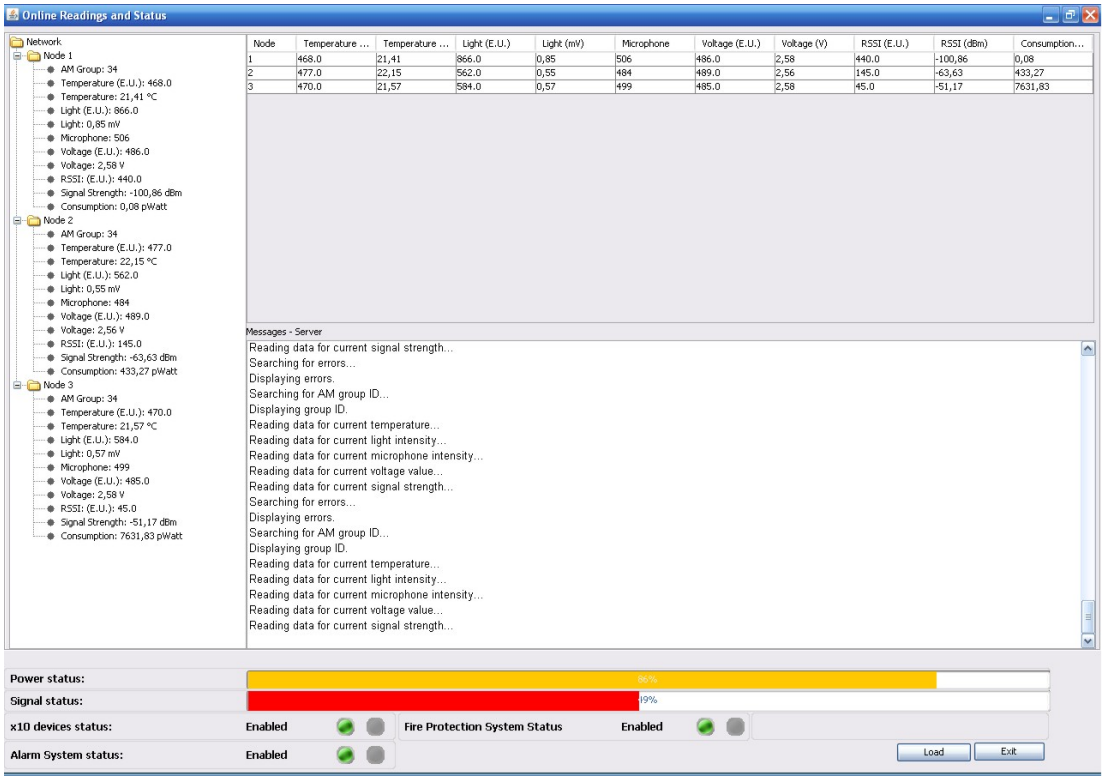


Figure 34: The WiSecurity main screen

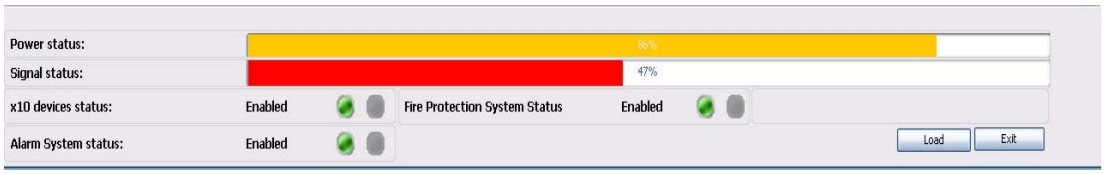


Figure 35:WiSecurity status labels

The WiSecurity system is composed from one hundred forty six (146) classes written in Java, integrating 15,100 lines of source code. For the implementation of the system three free available packages are included:

- The TinyOS 2.0 Java – based distribution (TinyOS, 2006). The SerialForwarder application coming from the TinyOS 2.0 distribution and its subcomponents are used from the WiSecurity system. SerialForwarder acts as a server for the WSN. It actually enables the concurrent listening for packets from one or more clients at the same time. Regarding the WiSecurity, SerialFowarder, as the name implies, acts as a forwarder of packets to the system,

- The Java X10 Project (TJX10P-13, 2006), is an x10 – Java API for integrating an x10 – based network with the WiSecurity System. The Java x10 Projects is an already implemented collection of classes, enabling the management of the x10 network under Java.
- The JavaMail API v. 1.4, provided from Sun Microsystems (2006. This API is used for email notification. It is used from the security and fire protection subsystem. An email notification, to one or two recipients can be sent when the alarm and/or sprinkler are turned on or off.

In addition, 813 lines form the source code in nesC which used for the Mica2 boards. The operating system which is selected for the sensor nodes was the TinyOS 2.0 while the compiler was the nesC compiler (NCC), version 1.2.1. It follows a short description for the most important classes of the WiSecurity system:

- **AllSenses class:** AllSenses is used to receive packets from the motes, regarding errors during sensing, group ID, row temperature value, sender for temperature, row temperature to celsius, row light value, sender for light, light to voltage, row microphone value, sender for microphone, row voltage value, sender for voltage, row voltage to voltage, row RSSI value (signal strength), sender for RSSI, RSSI to dBm, RRSI to watt (consumption). This class cannot be used directly but it can be executed via a cygwin bash.
- **AllSensesSend class**: AllSensesSend is used to send packets to the motes, regarding the sensing interval and the leds status (on | off). This class cannot be used directly but it can be executed via a cygwin bash.
- **AlarmSystem class**: AlarmSystem, loads the input from AllSenses class using a cygwin shell script on the background. It manages this input separately determining any potential threat including sound and fire.
- **ClientSSLSocket class**: Description: Manages the connection from the client side to the server side based on SSL Sockets.
- **CygwinExecuteCommands class**: CygwinExecuteCommands, provide a seamless - background execution process for compilation of nesc code (Tested for the TinyOS 2.0 distribution).

- **CygwinJavaApps class**: CygwinJavaApps creates a seamless cygwin based connection. It searches the registry for the java and cygwin path. It optimizes (according the Unix syntax) the applied commands. Finally uses an input stream reader for reading the input. At the very end, appends the input from the stream reader on a text area.
- **CygwinRunCommands class**: CygwinRunCommands provides a seamless - background execution of any command that can be executed on the cygwin regarding the TinyOS 2.0 distribution or not.
- **CygwinUploadToMote class**: CygwinUploadToMote provide a seamless - background execution process for uploading nesc code on motes (Tested for the TinyOS 2.0 distribution). It can be set according to the mote Id, the sensorboard, the communication board etc.
- **DisApp class**: DisApp extends frame and is part of the Dis application. It manages connections with the server while is acting as a middleware for the Wireless Sensor Network and the x10 network using an SSL encrypted connection for each communication with the server.
- **DisAppServer class**: DisAppServer manages connections with the clients, enables/ disables the x10 network, enables/disables the Alarm and Fire Protection System, acts as a middleware for the WSN and the x10 network.
- **DisAppServerOptions class**: extends frame. It corporate settings regarding the server, the x10 interface module, fire and alarm system settings and email notification.
- **EmailNotification class**: EmailNotification **is** used for sending alerts or errors from the system to a one or two recipients. This class is using the Java Mail API 1.4.
- **SenseData class**: SenseData extends frame library and it is part of the server. It is used for displaying real time data from the WSN. At the same time informs the user for current system status, for the alarm protection and for the fire protection system status.
- **SensorsData class:** SensorsData manipulates the input data from the Wireless Sensor Network. It calls on the background the cygwin and runs the

AllSenses class. It holds a while loop with a switch statement for converting each packet coming from the network to meaningful data for the application.

- **SerialForwarder class**: SerialForwarder uses a thread for a background call of cygwin. It actually runs the SerialForwarder GUI provided from TinyOS 2.0 distribution.
- **x10Server class**: x10Server is an integral part for the WiSecurity and fire protection system. It implements communication with x10 devices. These devices are slightly integrated with the system. This class act as a server implementing x10 based commands, like open/close individual devices, open/close all devices, open/close appliance modules.
- **AllSensesC** (nesC module) and **AllSensesAppC** (nesC configuration): AllSensesC and its configurationAllSensesAppC sense the environment where it is deployed and gather data based on the MTS300 sensorboard capabilities. Such data, includes Voltage measurement, strength of signal, additionally it covers temperature/light/Microphone sensing features. AllSensors uses the collection layer provided by TinyOS 2.0 distribution. It assumes as the root node the node with the id 1. After uploading on motes, a user is able, over the air to change the interval sensing ratio used from timers and the leds status to ON or OFF. The current implementation is based on the MTS300CB sensor board.

3.5. A security and fire protection system; case study

The WiSecurity system combines a sensor network and an x10 network integrating into the same level two communication interfaces. For the WSN deployment we use three MPR400CB sensor boards (Figure 37) attached on a MICA2 processor (Figure 36) and an RS – 232 PC interface (MIB510CA) purchased from Crossbow Inc (Figure 38).

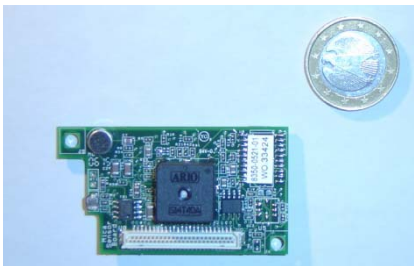


Figure 37: MPR400CB Sensor board

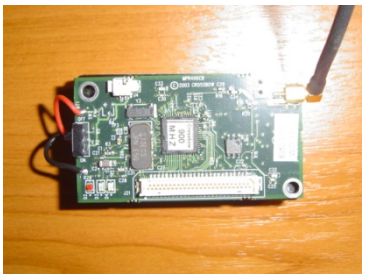


Figure 36: Mica2 mode

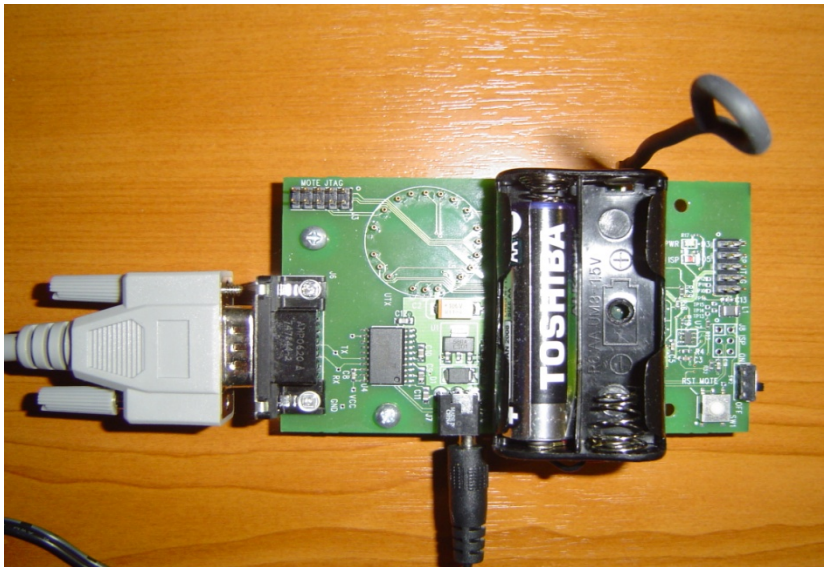


Figure 38: The RS - 232 communication interface

The first phase was to upload nesC code on the nodes. Using the MIB510CA interface, via a serial port, we upload on each of the three modes a TinyOS – based application. This application scope was to allow modes to collect information from their environment and forward it to the next hop node. The last in hierarchy node

operates as a forwarder to the MIB510CA sensor board (named as sink). On the second level we attach two AA alkaline batteries on each node for power supply. The last step was to attach the MIB510CA interface on a Linux – based PC and initialize the WiSecurity system. In order for the WiSecurity system to operate as appropriate we set up the TinyOS 2.0 distribution and the Java Runtime Environment (JRE) on the end machine.

Both the TinyOS distribution and the JRE environment, have to operate successfully before the WiSecurity system integration. The next step was to deploy randomly the modes on an indoor environment and turn them on. On the end system (where actually the WiSecurity system is deployed) we use the SerialForwarder TinyOS – based application in order to collect network packets. In this way, we are able to observe any incoming row data units from the x10 network.

Next, we attach the x10 communication board for the x10 computer interface (CM11) (Figure 39). We attach the CM11 device on a USB port on an end machine where the WiSecurity is installed. We apply the check mechanism in order to ensure that our system was able to communicate with the x10 computer interface. After that we specify, using the WiSecurity system, the address of the alarm and the sprinkler device (Figure 40). Both the alarm and the sprinkler device are attached on an x10 light module. During the testing period we replace the alarm device with a lamp.

Figure 39: CM11 interface

Figure 40: A lamp and an appliance module

Using the option properties, provided from the WiSecurity system we apply the desired security level for our application (Figure 41). The available levels were high, medium and low security. WiSecurity, is preconfigured with the above three security levels. Each level holds a maximum, medium and low threshold value. Such values are row data arrive from the WSN, including temperature, light intention and sound. Using an autonomous decision algorithm, WiSecurity compares incoming values against the security level which described above. Flow charts 31 and 32 describe the way that WiSecurity acts in case of light and sound detection and/or temperature fluctuations. On an event, WiSecurity triggers an x10 device to turn on. Using an internal counter, WiSecurity is able to turn off a device after a period of time.

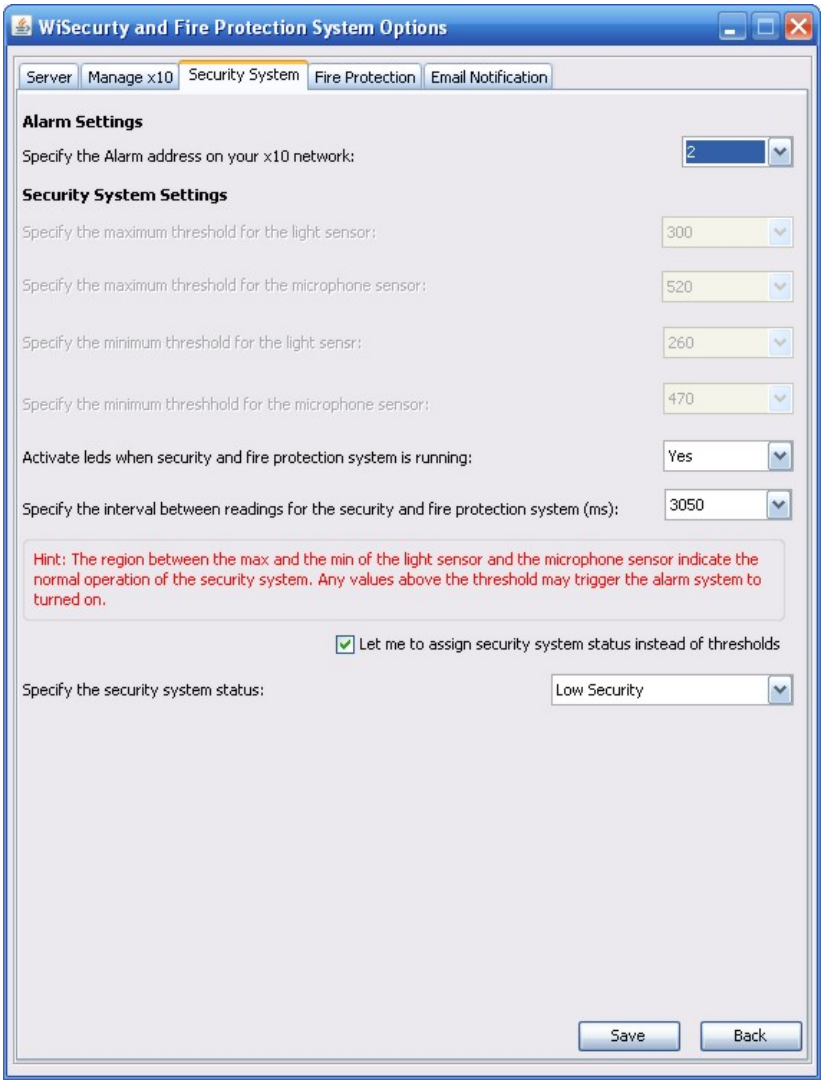

WiSecurty and Fire Protection System Options
Server
Manage x10
Security System
Fire Protection
Email Notification
Alarm Settings
Specify the Alarm address on your x10 network:
2
Security System Settings
Specify the maximum threshold for the light sensor:
300
Specify the maximum threshold for the microphone sensor:
520
Specify the minimum threshold for the light sensr:
260
Specify the minimum threshhold for the microphone sensor:
470
Activate leds when security and fire protection system is running:
Yes
Specify the interval between readings for the security and fire protection system (ms):
3050
Hint: The region between the max and the min of the light sensor and the microphone sensor indicate the normal operation of the security system. Any values above the threshold may trigger the alarm system to turned on.
Let me to assign security system status instead of thresholds
Specify the security system status:
Low Security
Save
Back

Figure 41: WiSecurity security level options

In parallel, user is able to manage both the sensor network and the x10 network remotely. A remote user is able to apply x10 commands directly to the x10 network without any other in system interaction (Figure 42). Additionally, he is able to collect information from the sensor network and advertise commands. Communication between the two ends is performed upon a secure communication channel.

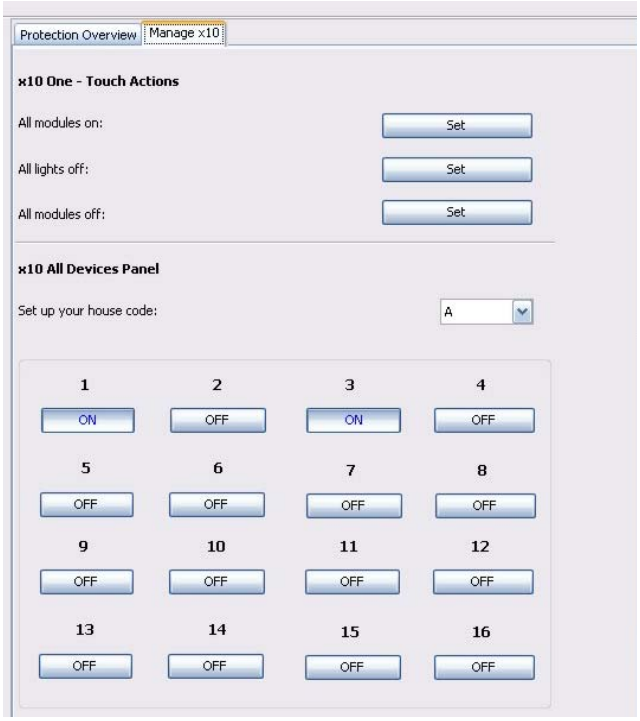


Figure 42: x10 network management UI

3.6. Resources

WiSecurity system is not a simulator for WSN. For the implementation and the testing face is used a series of hardware devices. These hardware devices are:

a. **Hardware devices used for the WSN deployment**, including:

- An RS – 232 PC Interface board (MIB510CA),
- Four Mica2 processor/ radio boards (MPR400CB),
- Three light/ temperature/ acoustic sensor boards (MTS300CB),
- One USB – to – Serial converter,

b. **Hardware devices used for the x10 network deployment**, including:

- An USB Computer Interface (CM11),

- Two Plug-in Lamp Modules (LM12),
- One Transceiver module (TM13),
- One Plug-in Appliance Module (AM12),
- One Socket Rocket Lamp Module (LM15).

The devices which are listed for the WSN deployment are purchased from Crossbow Inc., a company located in the United States of America. The devices listed for the x10 network deployment, are purchased from IntelliHome Corporation. IntelliHome, is the main distributor of x10 – based devices in Europe. IntelliHome is located in Belgium.

The cost for the devices which are used for the testing and implementation phase of the WiSecurity system, are listed below on Table 36.

	WSN devices (from Crossbow Inc.)	***x10 devices (from IntelliHome Corporation)***
Partial Cost (€)	1.6511,37 €	169 €
Total Cost (€)	**1.820,37 €**	

Table 4: WiSecurity equipment cost

Bibliography

[1] Crossbow Technology, 2006a. MTS/MDA Sensor Board Users Manual. [online].

Available from: http://www.xbow.com/Support/Support_pdf_files /MTS-MDA_Series_Users_Manual.pdf [Accessed 5 February 2010].

[2] Crossbow Technology, 2006b. MPR-MIB Wireless Module Users Manual. [online].

Available from: http://www.xbow.com/Support/Support_pdf_ files/MPR-MIB_Series_Users_Manual.pdf [Accessed 5 February 2010].

[3] Sun Microsystems, 2006. JavaMail API 1.4. [online]. Available from: http://java.sun.com/products/javamail/javadocs/index.html [Accessed 19 December 2006].

[4] TJX10P-13, 2006. The Java X10 Project. [online]. Available from: http://x10.homelinux.org/ [Accessed 5 February 2010].

Chapter 4. Conclusion

WiSecurity system is a multi - application middleware for WSNs. WiSecurity fulfils a collection of activities which are the following:

i. An easy to use cygwin and TinyOS – based application, enabling the management of the sensors hardware.
ii. An autonomous security and fire protection system, which is based on the data which is collected from the WSN.
iii. Integration of an x10 network.
iv. An x10 control application, enabling the management of the x10 network.
v. A remote management application of both the WSN and the x10 network.
vi. Over the air commands for the WSN.

WiSecurity improves the capabilities of current available middleware applications for WSNs. It uses heterogeneous network architectures, including TCP/ IP, WSN and x10 in a way that is feasible to export the WSN collected data to an intelligent decision making system. This decision making system integrates a powerful security and fire protection system which is able to react with an x10 network.

Further research, may include the integration of a powerful power management schema for the WSN. Such schema should eliminate at the minimum the power consumption for the sensor nodes.

APPENDIX A. Use Case Description

Use case	**Start Protection System**	
Primary Actors	User (server side)	
Preconditions	The SerialForwarder App should already run, the USB – to Serial x10 interface module should be connected with the system. Additionally, the interface board should be attached on the computer.	
Trigger	The initialization of the x10 control, security and fire subsystems	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User activates the protection system	2. System checks if the USB – to Serial x10 interface module and sensor board for the WSN subsystem are properly connected.
		3. System uses two icons on screen, displaying the user that the x10 Control, security and fire protection subsystems started successfully.

Table 5: Use Case: Start Protection System

Use case	**Protection System doesn`t start**

Primary Actors	User (server side)	
Preconditions	The SerialForwarder App should already run, the USB – to Serial x10 interface module should be connected with the system. Additionally, the interface board should be attached on the computer.	
Trigger	Extension of Start Protection System use case	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User activates the protection system	2. System checks if the USB – to Serial x10 interface module and sensor board for the WSN subsystem are properly connected.
		3. System could not locate the USB – to Serial x10 interface module.
		4. User is informed for the problem.
		5. System could not communicate with the SeralForwarder.
		6. User is informed for the problem.

Table 6: Use Case: Protection System doesn`t start

Use case	**Stop protection system**	
Primary Actors	User (server side)	
Preconditions	The protection system should already running.	
Trigger	Disables the security and fire subsystems	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User deactivates the protection system	2. System stops the x10 control, security and fire subsystems.
		3. System disables the two icons on screen, for showing in such way to the user that the x10 control, security and fire protection subsystems are successfully shut down.

Table 7: Use Case: Stop protection system

Use case	**Run Java Application**
Primary Actors	i. User (server side), ii. User (client side)
Preconditions	The cygwin package and the TinyOS 2.0 distribution have to be already installed on the computer.
Trigger	Run a Java – based application, using the cygwin package on the background.

Typical Course Of Events	Actor Action	System Response
	1. User specifies the location of the directory where the Java class is.	2. System initializes on the background the cygwin package and run the application. At the end displays the results to the user.

Table 8: Use Case: Run Java Application

Use case	**Upload to mote**	
Primary Actors	i. User (server side), ii. User (client side)	
Preconditions	The cygwin package and the TinyOS 2.0 distribution have to be already installed on the computer. Additionally, the interface board should be attached on the computer.	
Trigger	Compiles or Recompiles nesC code and uploads it on a selected node.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the location of the directory where the nesC code is.	
	2. User specifies additional requirements for	2. System initializes on the background the

	uploading.	cygwin package and run the appropriate commands, uploads the compiled application on the node and finally displays the results to the user.

Table 9: Use Case: Upload to mote

Use case	**Compile**	
Primary Actors	i. User (server side), ii. User (client side)	
Preconditions	The cygwin package and the TinyOS 2.0 distribution have to be already installed on the computer.	
Trigger	Compiles nesC code.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the location of the directory where the nesC code is.	
	2. User specifies additional requirements for compiling.	2. System initializes on the background the cygwin package and run the appropriate commands, compiles the nesC code and finally displays the results to the

	user.

Table 10: Use Case: Compile

Use case	**Run commands**	
Primary Actors	i. User (server side), ii. User (client side)	
Preconditions	The cygwin package and the TinyOS 2.0 distribution have to be already installed on the computer.	
Trigger	Runs any available command which is cygwin and TinyOS 2.0 compatible.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies a command.	2. System initializes on the background the cygwin package and run the appropriate command, finally displays the results to the user.

Table 11: Use Case: Run commands

Use case	**Read from the WSN**
Primary Actors	i. User (server side), ii. User (client side)
Preconditions	The cygwin package and the TinyOS 2.0 distribution

	have to be already installed on the computer. Additionally, the interface board should be attached on the computer.	
Trigger	Collects data from the WSN subsystem.	
Typical Course Of Events	**Actor Action**	**System Response**
		1. System initializes on the background the cygwin package and starts reading data from the WSN.
		2. System displays the collected data to the user.
Steps 1 and 2 can be run on an infinite loop.		

Table 12: Use Case: Read from the WSN

Use case	**Manage Server**	
Primary Actors	User (server side)	
Preconditions	No preconditions	
Trigger	Set the local server port and the maximum number of concurrent connections for the WiSecurity server.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the port	

	on which the local server listens.	
	2. User specifies the maximum permitted concurrent connections.	3. The system, stores the user `s preferences on the local disk

Table 13: Use Case: Manage Server

Use case	**Manage x10 server**	
Primary Actors	User (server side)	
Preconditions	The USB – to Serial x10 interface module has to be properly connected.	
Trigger	Sets the x10 interface module communication port, the x10 network address and a list of devices which will be turned on with the alarm device.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the communication port of the x10 interface module.	
	2. User specifies the x10 network address.	
	3. User specifies a list of devices.	4. The system, stores the user `s preferences on the local disk

Table 14: Use Case: Manage x10 server

Use case	**Manage Fire subsystem**	
Primary Actors	User (server side)	
Preconditions	The USB – to Serial x10 interface module has to be properly connected.	
Trigger	Sets the x10 – based device code for the sprinkler and the maximum allowed temperature value for the fire protection subsystem.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the code address of the x10 – based device.	
	2. User specifies the x10 network address.	
	3. User specifies the maximum temperature value.	4. The system stores the user `s preferences on the local disk

Table 15: Use Case: Manage Fire subsystem

Use case	**Manage email**	
Primary Actors	User (server side)	
Preconditions	No preconditions	
Trigger	Set the email settings of an email account.	
Typical Course Of	**Actor Action**	**System Response**

Events		
	1. User specifies the sender email, one or two recipients, a subject, the account type, the outgoing mail server host name, the username and the password for access on the outgoing mail server.	2. The system stores the user `s preferences on the local disk

Table 16: Use Case: Manage email

Use case	**Manage Security subsystem**	
Primary Actors	User (server side)	
Preconditions	No preconditions	
Trigger	Sets the code address of the alarm device, the maximum and minimum allowed limits for light and sound detection.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the code address of the alarm.	
	2. User specifies the maximum and minimum allowed limits for light and sound detection	

	3. User specifies the leds and the interval status	4. The system stores the user `s preferences on the local disk

Table 17: Use Case: Manage Security subsystem

Use case	**Start local server**	
Primary Actors	User (server side)	
Preconditions	No preconditions	
Trigger	Initializes the WiSecurity server.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User enables the local server.	2. System starts to listen from new connections.

Table 18: Use Case: Start local server

Use case	**Stop local server**	
Primary Actors	User (server side)	
Preconditions	No preconditions	
Trigger	Stop the WiSecurity server.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User disables the local server.	2. System stops to listen from new connections.

Table 19: Use Case: Stop local server

Use case	**Manage Validation**	
Primary Actors	User (client side)	
Preconditions	The client should already handle the required key for the authentication on the local disk.	
Trigger	User is connected with the WiSecurity system and its subsystems	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User enters the secret password	2. System checks the integrity of the password and forces a connection with the server side.
		3. System responds to the client's request.
		4. System initializes a connection with the WSN subsystem and transfers, in an infinite loop, the incoming data.

Table 20: Use Case: Manage Validation

Use case	**Non valid registration key**
Primary Actors	User (client side)
Preconditions	The client should already handle the required key for

	the authentication on the disk.	
Trigger	Extension of Manage Validation use case	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User enters the secret password	2. System checks the integrity of the password.
		3. System informs the user that the connection, to the server side, could not be established.

Table 21: Use Case: Non valid registration key

Use case	**Turn an individual x10 device on**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side.	
Trigger	Enables the control of an individual device on the x10 network.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the address of the device.	
	2. User specifies the code of the device.	3. System sends a request to the server side.

		4. The server forwards the request to the x10 network. And returns the respond to the client.
		5. System informs the user for the result of his command (Success/ Fail)

Table 22: Use Case: Turn an individual x10 device on

Use case	**Turn an individual x10 device off**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side.	
Trigger	Disables an individual device on the x10 network.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the code of the device.	2. System sends a request to the server side.
		3. The server forwards the request to the x10 network. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)

Table 23: Use Case: Turn an individual x10 device off

Use case	**Turn all lights off**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side.	
Trigger	Close all the light modules on the x10 network.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the address of the x10 network.	2. System sends a request to the server side.
		3. The server forwards the request to the x10 network. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)

Table 24: Use Case: Turn all lights off

Use case	**Turn all modules off**
Primary Actors	User (client side)
Preconditions	The client should be already connected with the server side.
Trigger	Disables all the modules (including light, socket and

	appliance modules) on the x10 network.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User specifies the address of the x10 network.	2. System sends a request to the server side.
		3. The server forwards the request to the x10 network. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)

Table 25: Use Case: Turn all modules off

Use case	**Turn all modules on**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side.	
Trigger	Enables all the modules (including light, socket and appliance modules) on the x10 network.	
Typical Course Of Events	**Actor Action**	**System Response**

	1. User specifies the address of the x10 network.	2. System sends a request to the server side.
		3. The server forwards the request to the x10 network. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)

Table 26: Use Case: Turn all modules on

Use case	**Turn on sprinkler**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side.	
Trigger	Enables only the sprinkler device, system uses the already stored address and code of the sprinkler device. The stored information is on the server side.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User sends a request, for enabling the sprinkler device.	2. System sends a request to the server side.

		3. The server forwards the request to the x10 network. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)
		5. System turns off the sprinkler, automatically after 5 minutes.

Table 27: Use Case: Turn on sprinkler

Use case	**Turn off sprinkler**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side.	
Trigger	Disables only the sprinkler device, system uses the already stored address and code of the sprinkler device. The stored information is on the server side.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User sends a request, for disabling the sprinkler device.	2. System sends a request to the server side.
		3. The server forwards the request to the x10

		network. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)

Table 28: Use Case: Turn off sprinkler

Use case	**Turn on alarm**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side.	
Trigger	Enables only the alarm device, system uses the already stored address and code of the alarm device. The stored information is on the server side.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User sends a request, for enabling the alarm device.	2. System sends a request to the server side.
		3. The server forwards the request to the x10 network. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)

		5. System turns off the alarm, automatically after 5 minutes.

Table 29: Use Case: Turn on alarm

Use case	**Turn off alarm**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side.	
Trigger	Disables only the alarm device, system uses the already stored address and code of the sprinkler device. The stored information is on the server side.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User sends a request, for disabling the alarm device.	2. System sends a request to the server side.
		3. The server forwards the request to the x10 network. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)

Table 30: Use Case: Turn off alarm

Use case	**Start Security subsystem**

Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side and the security subsystem to be already disabled on the server side or on the client side.	
Trigger	Initializes the security subsystem, on the server side	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User sends a request, for enabling the security subsystem.	2. System sends a request to the server side.
		3. The server applies the request and sends a respond to the client.
		4. System informs the user for the result (Success/ Fail)

Table 31: Use Case: Start Security subsystem

Use case	**Start Fire subsystem**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side and the fire subsystem to be disabled on the server side or on the client side.	
Trigger	Initializes the fire subsystem, on the server side	
Typical Course Of	**Actor Action**	**System Response**

Events		
	1. User sends a request, for enabling the fire subsystem.	2. System sends a request to the server side.
		3. The server applies the request and sends a respond to the client.
		4. System informs the user for the result (Success/ Fail)

Table 32: Use Case: Start Fire subsystem

Use case	**Stop Security subsystem**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side and the security subsystem to be already activated either on the server side or on the client side.	
Trigger	Disables the security subsystem, on the server side	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User sends a request, for disabling the security subsystem.	2. System sends a request to the server side.
		3. The server applies the request and sends a

		respond to the client.
		4. System informs the user for the result (Success/ Fail)

Table 33: Use Case: Stop Security subsystem

Use case	**Stop Fire subsystem**	
Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side and the fire subsystem to be already activated either on the server side or on the client side.	
Trigger	Disables the fire subsystem, on the server side	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User sends a request, for disabling the fire subsystem.	2. System sends a request to the server side.
		3. The server applies the request and sends a respond to the client.
		4. System informs the user for the result (Success/ Fail)

Table 34: Use Case: Stop Fire subsystem

Use case	**Manage interval**

Primary Actors	User (client side)	
Preconditions	The client should be already connected with the server side. The SerialForwarder application should run (on the server side) and the WSN subsystem to be activated.	
Trigger	Enables the user to change, remotely, the interval between the sensing periods on the WSN.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User sends a request, with the desired interval	2. System sends a request to the server side.
		3. The server, forward the request to the WSN subsystem. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)

Table 35: Use Case: Manage interval

Use case	**Manage leds**
Primary Actors	User (client side)
Preconditions	The client should be already connected with the server side. The SerialForwarder application should run (on the server side) and the WSN subsystem to be activated.

Trigger	Enables the user to change, remotely, the status of the leds on the nodes. He is able to turn on or off the leds.	
Typical Course Of Events	**Actor Action**	**System Response**
	1. User sends a request, with the desired leds status.	2. System sends a request to the server side.
		3. The server forwards the request to the WSN subsystem. And returns the respond to the client.
		4. System informs the user for the result of his command (Success/ Fail)

Table 36: Use Case: Manage leds

APPENDIX B. FORMULAS FOR UNITS CONVERSION

B1. Converting temperature units into Celsius (Crossbow, Inc., 2006a)

ADC_FS = 1023,

ADC, is the returned value from ADC1,

R1 = 10000 Ω,

a = 0.00130705,
b = 0.000214381,
c = 0.000000093,

Rthr = (R1 * (ADC_FS - ADC)) / ADC

celcious = 1 / (a + b * Math.log(Rthr) + c * Math.pow(Math.log(Rthr), 3))

celsius - = 273.15

B2. Converting voltage units into mVolt

mVB, is the returned value from the ADC1

mV = mVB / 1024

B3. Calculating Battery Voltage (Octave Technology, 2006)

voltageSnif , is the returned value from pin A5

voltageRD = (1252353 / voltageSnif) / 1000;

B4. Calculating RSSI in dBm (Crossbow, Inc., 2006b)

BVolt, is the returned value from pin A5,

RSSIVal, is the returned value from ADC0

dBmFS = (BVolt * RSSIVal)/1024

dBmSS = -50 * dBmFS - 45.5

B5. Calculating RSSI in pWatt (Crossbow, Inc., 2006b)

dBmSS, is the output from the formula B4

convertToWatt = Math.pow(10, (dBmSS - 30) / 10)

dev = Math.pow(10, 12)

convertToWatt = convertToWatt * dev

B6. References

[1] Crossbow, 2006a. MTS/MDA Sensor Board Users Manual.

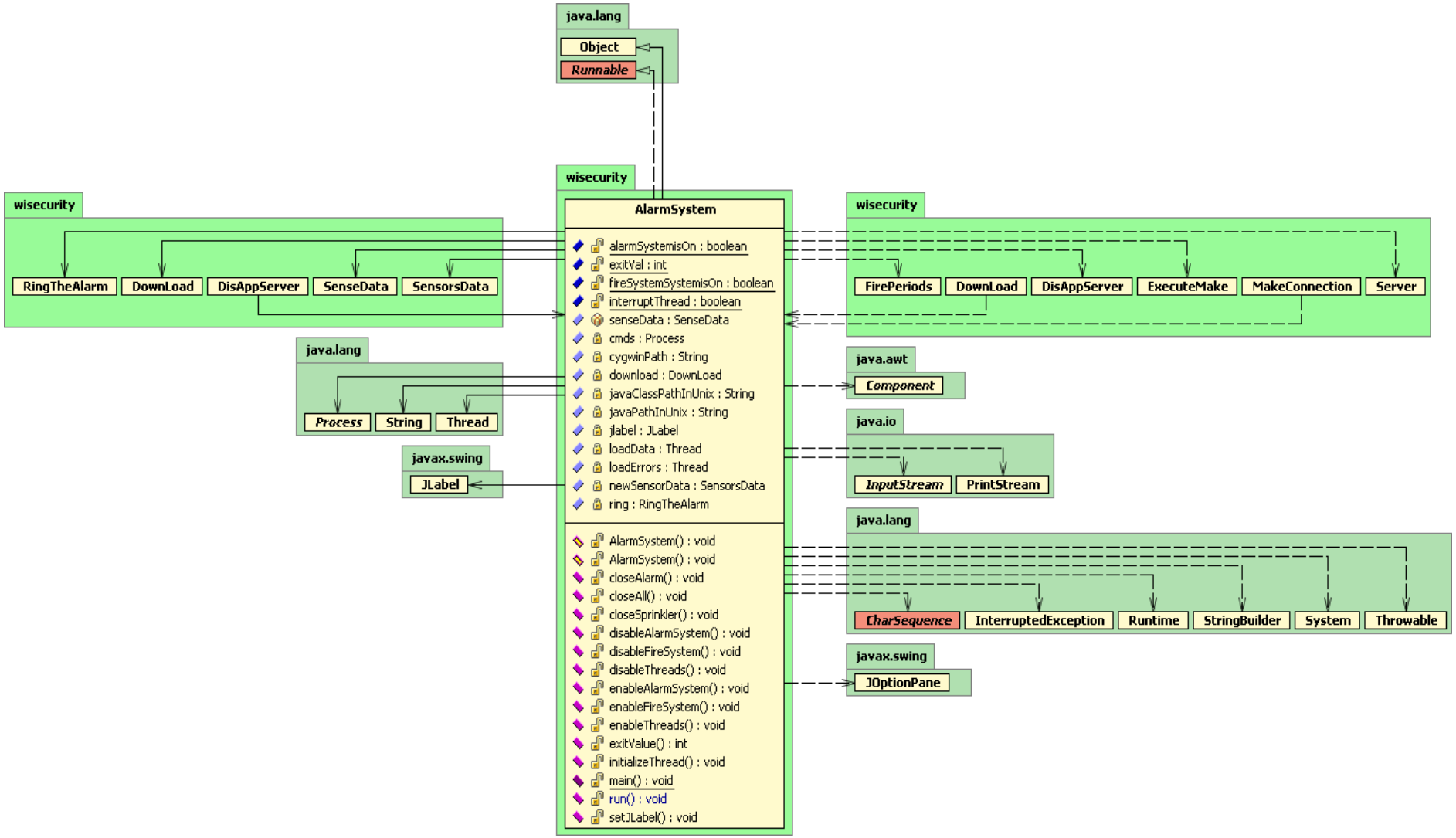
[2] Crossbow, 2006b. MPR - Mote Processor Radio Board MIB - Mote Interface / Programming Board User's Manual.

[3] THORN J., 2005. *Deciphering TinyOS Serial Packets*. Octave Technology.

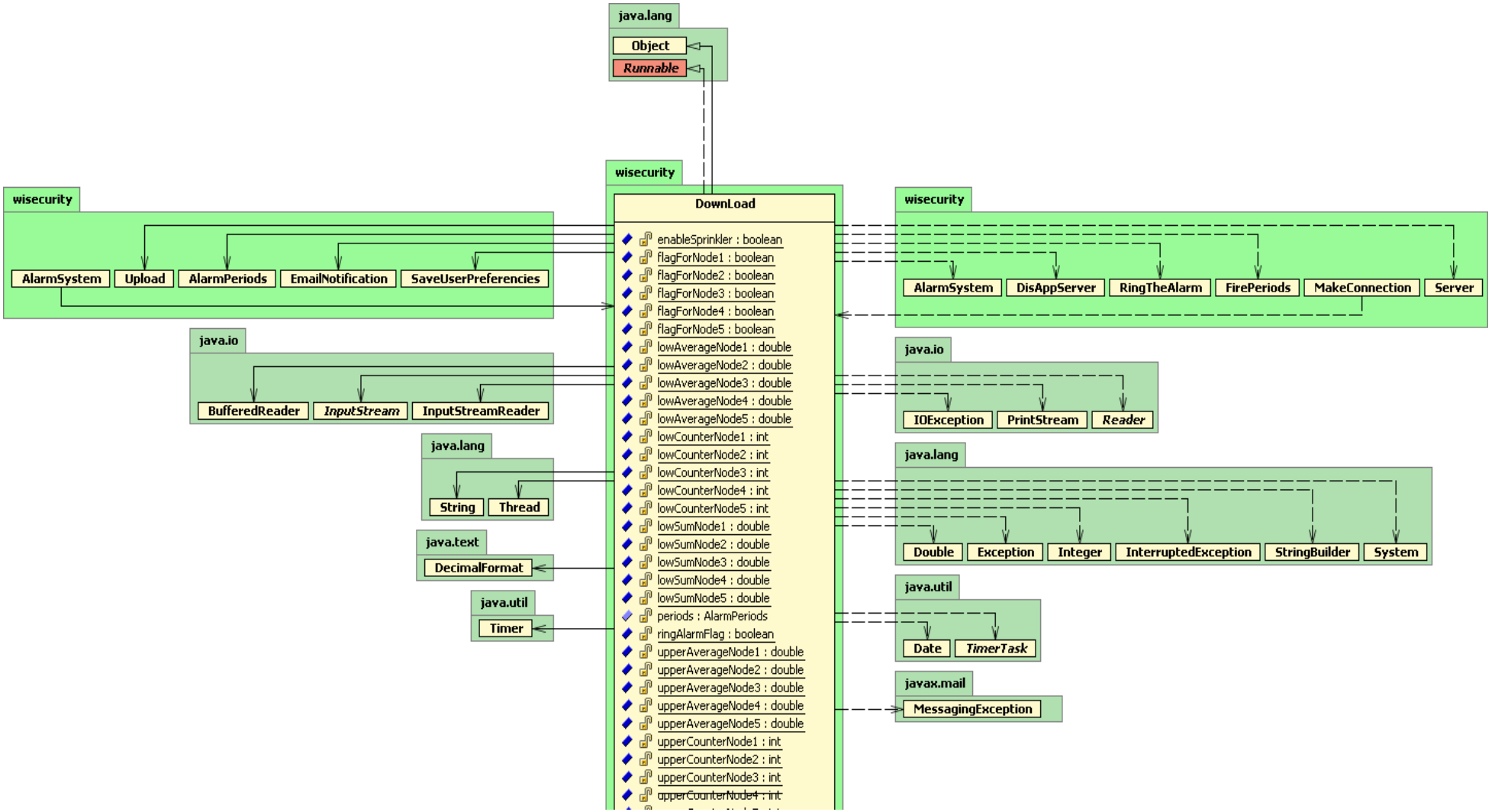
APPENDIX C. CLASS DIAGRAMS FOR THE WISECURITY SYSTEM

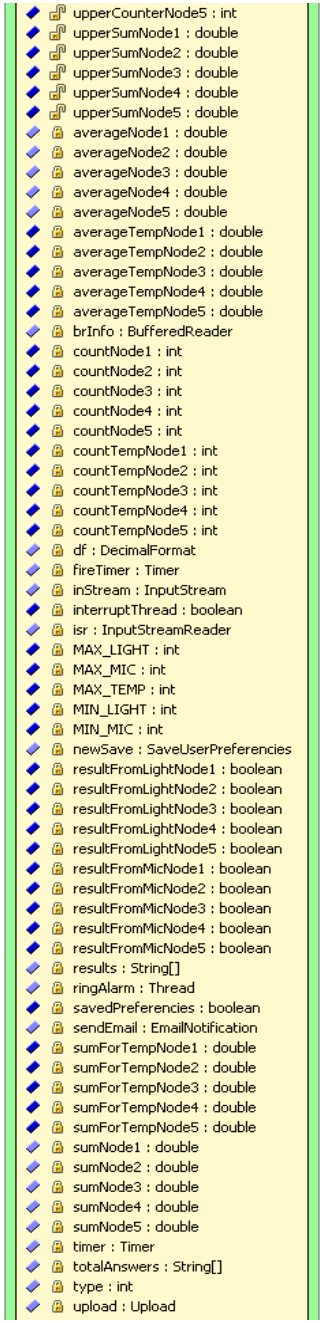
On this section, are selected only eighteen (18) of the one hundred forty six (146) classes from which the WiSecurity system is composed. These class diagrams depict the most important components for the WiSecurity system and its subsystems (including the x10 control, security and fire protection subsystems).

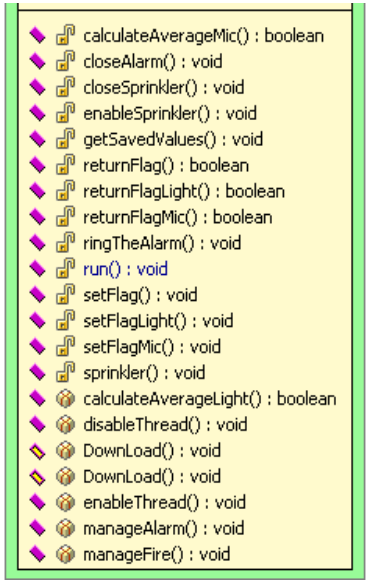
C1. The AlarmSystem class



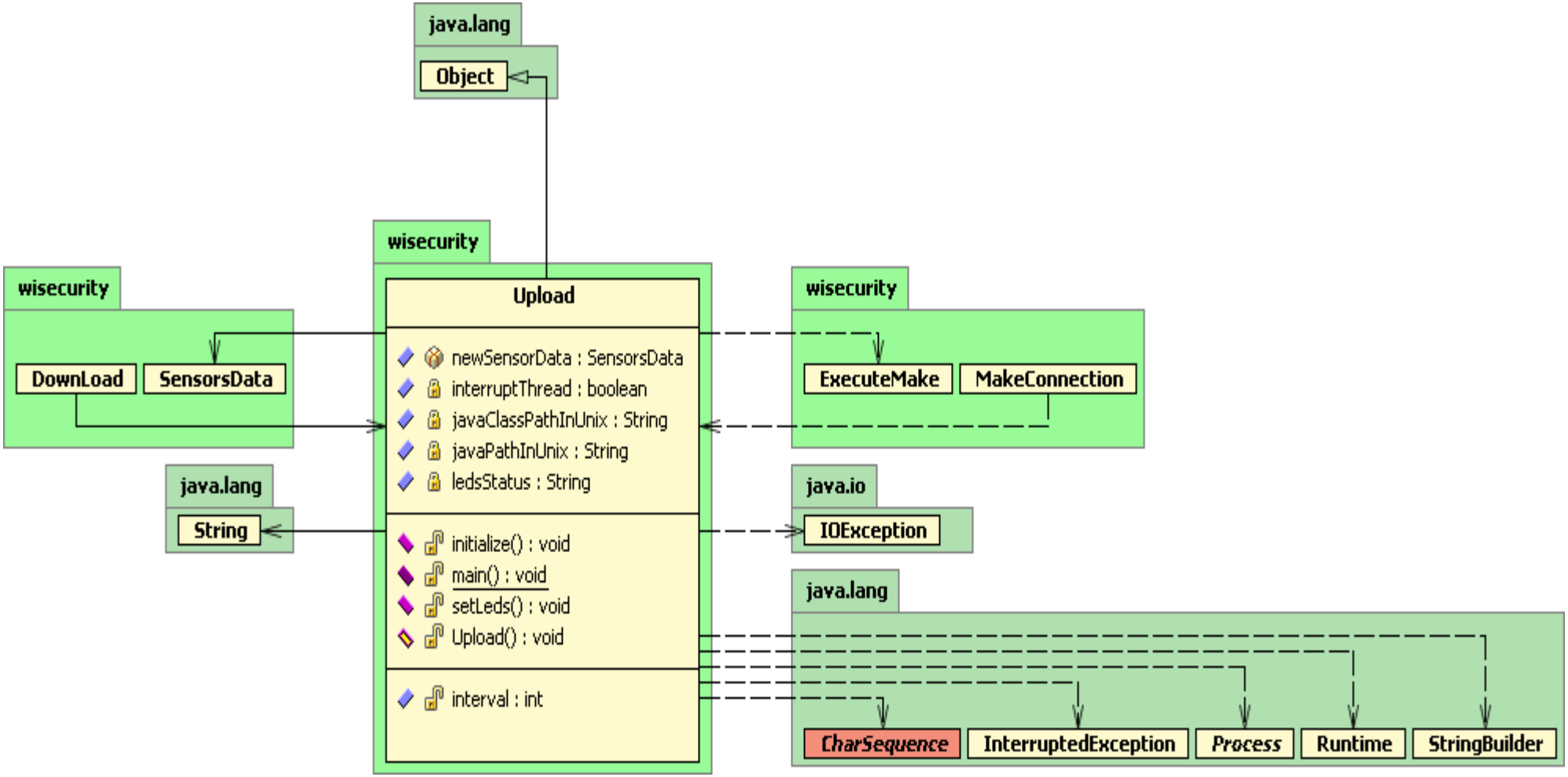
C2. The DownLoad class



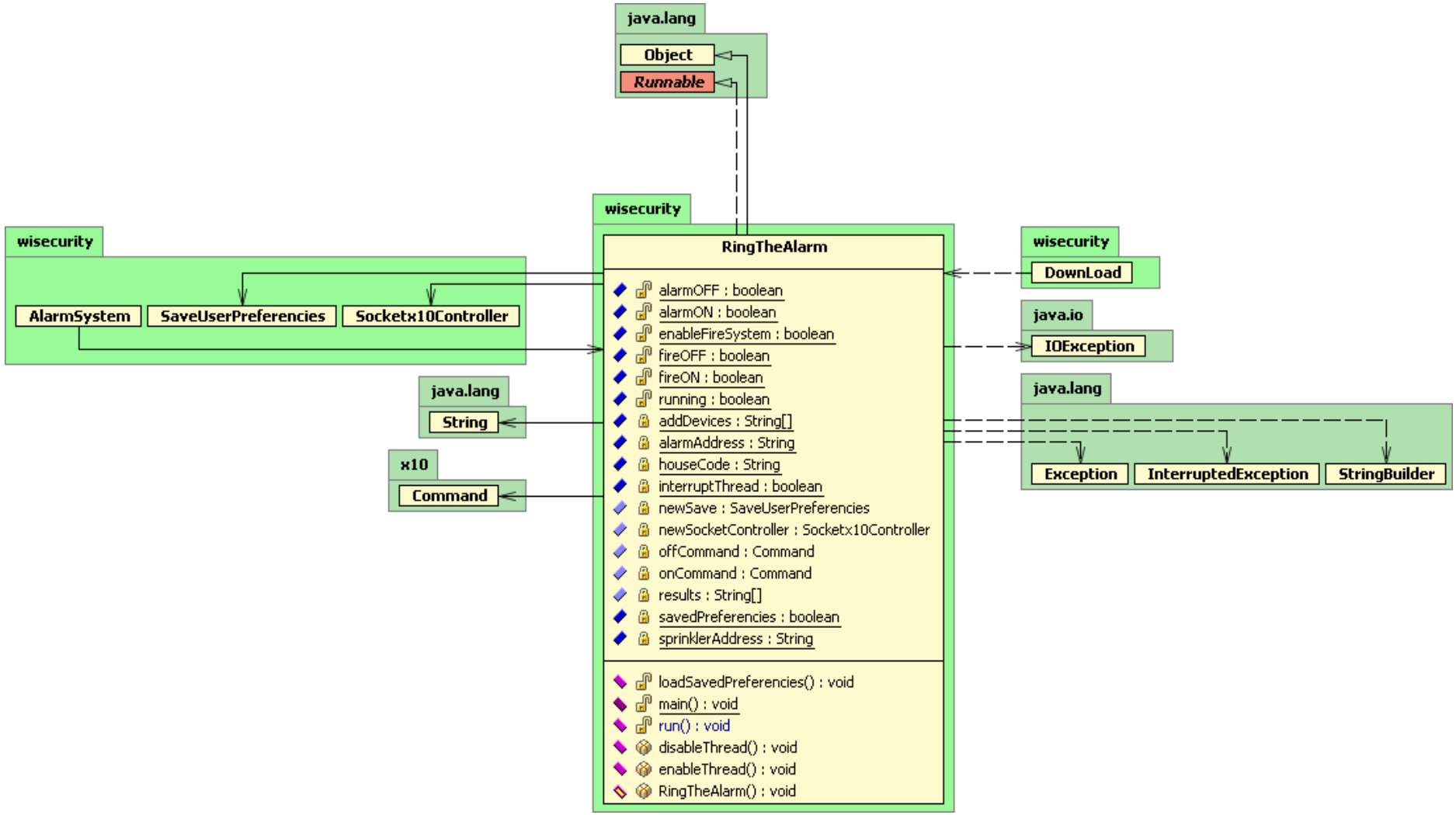

upperCounterNode5 : int
upperSumNode1 : double
upperSumNode2 : double
upperSumNode3 : double
upperSumNode4 : double
upperSumNode5 : double
averageNode1 : double
averageNode2 : double
averageNode3 : double
averageNode4 : double
averageNode5 : double
averageTempNode1 : double
averageTempNode2 : double
averageTempNode3 : double
averageTempNode4 : double
averageTempNode5 : double
brInfo : BufferedReader
countNode1 : int
countNode2 : int
countNode3 : int
countNode4 : int
countNode5 : int
countTempNode1 : int
countTempNode2 : int
countTempNode3 : int
countTempNode4 : int
countTempNode5 : int
df : DecimalFormat
fireTimer : Timer
inStream : InputStream
interruptThread : boolean
isr : InputStreamReader
MAX_LIGHT : int
MAX_MIC : int
MAX_TEMP : int
MIN_LIGHT : int
MIN_MIC : int
newSave : SaveUserPreferencies
resultFromLightNode1 : boolean
resultFromLightNode2 : boolean
resultFromLightNode3 : boolean
resultFromLightNode4 : boolean
resultFromLightNode5 : boolean
resultFromMicNode1 : boolean
resultFromMicNode2 : boolean
resultFromMicNode3 : boolean
resultFromMicNode4 : boolean
resultFromMicNode5 : boolean
results : String[]
ringAlarm : Thread
savedPreferencies : boolean
sendEmail : EmailNotification
sumForTempNode1 : double
sumForTempNode2 : double
sumForTempNode3 : double
sumForTempNode4 : double
sumForTempNode5 : double
sumNode1 : double
sumNode2 : double
sumNode3 : double
sumNode4 : double
sumNode5 : double
timer : Timer
totalAnswers : String[]
type : int
upload : Upload


calculateAverageMic() : boolean
closeAlarm() : void
closeSprinkler() : void
enableSprinkler() : void
getSavedValues() : void
returnFlag() : boolean
returnFlagLight() : boolean
returnFlagMic() : boolean
ringTheAlarm() : void
run() : void
setFlag() : void
setFlagLight() : void
setFlagMic() : void
sprinkler() : void
calculateAverageLight() : boolean
disableThread() : void
DownLoad() : void
DownLoad() : void
enableThread() : void
manageAlarm() : void
manageFire() : void

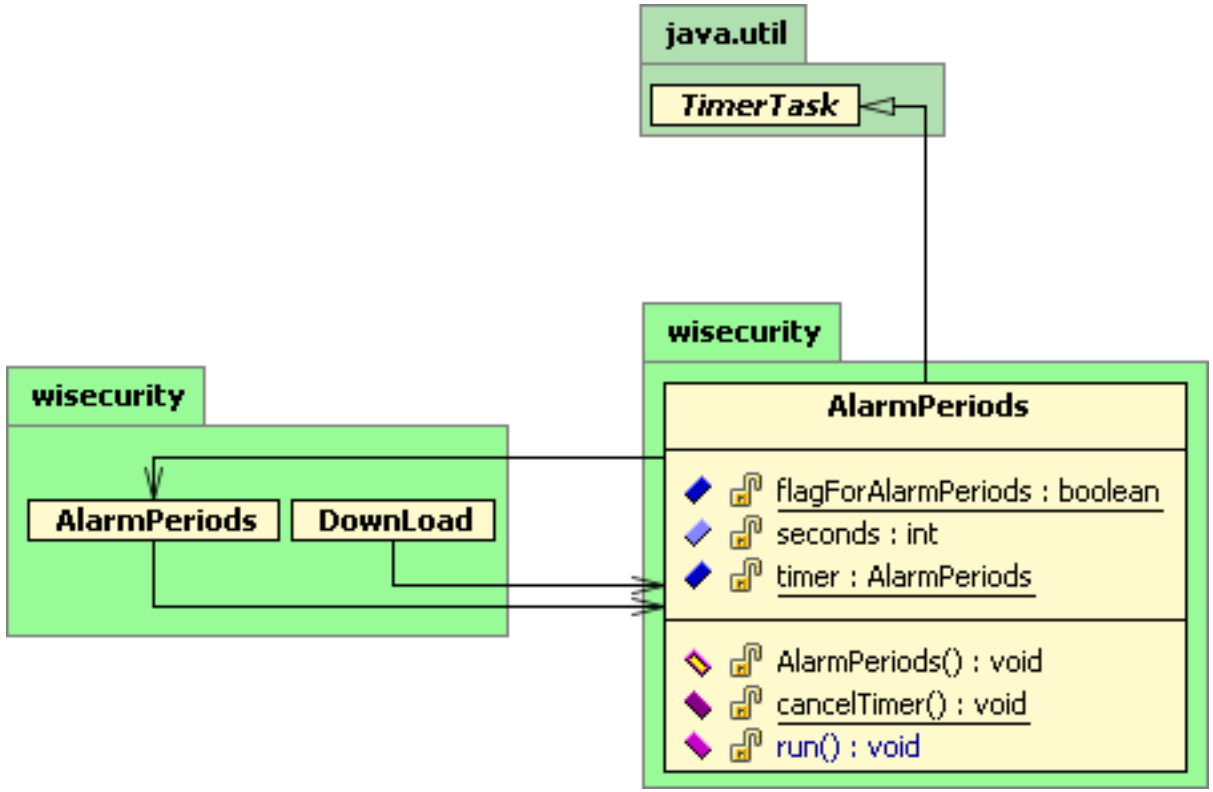
C3. The Upload class



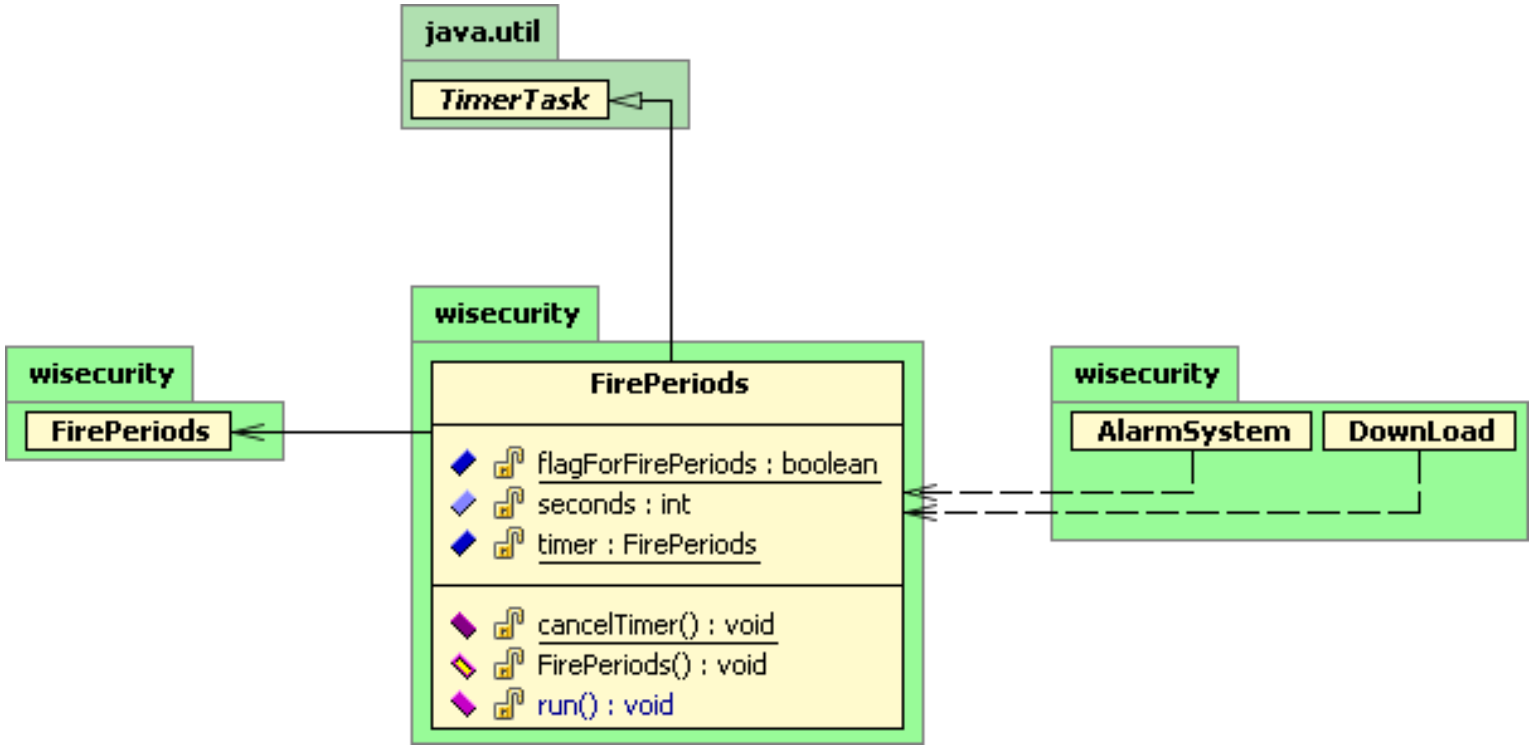
C4. The RingTheAlarm Class



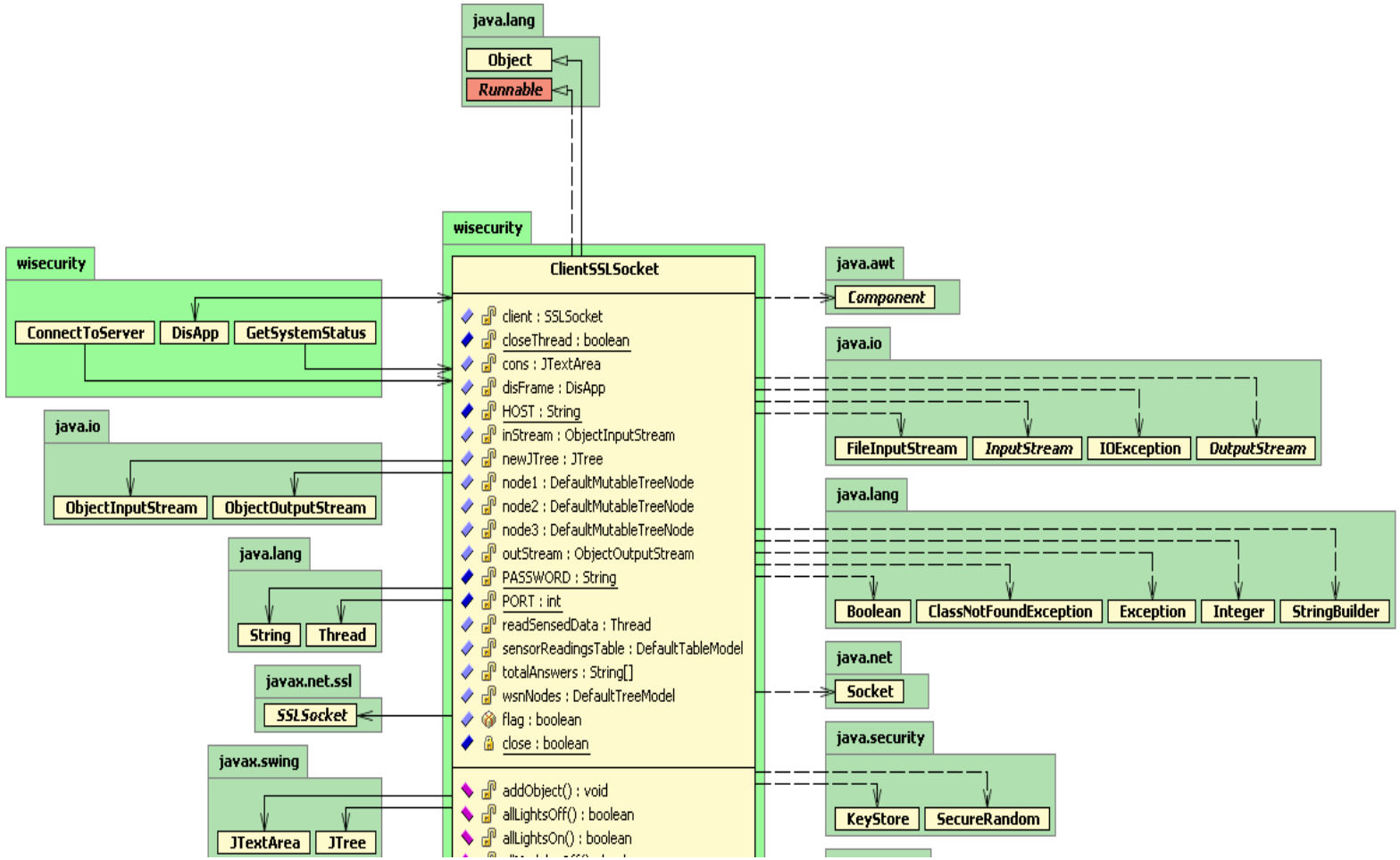
C5. The AlarmPeriods Class

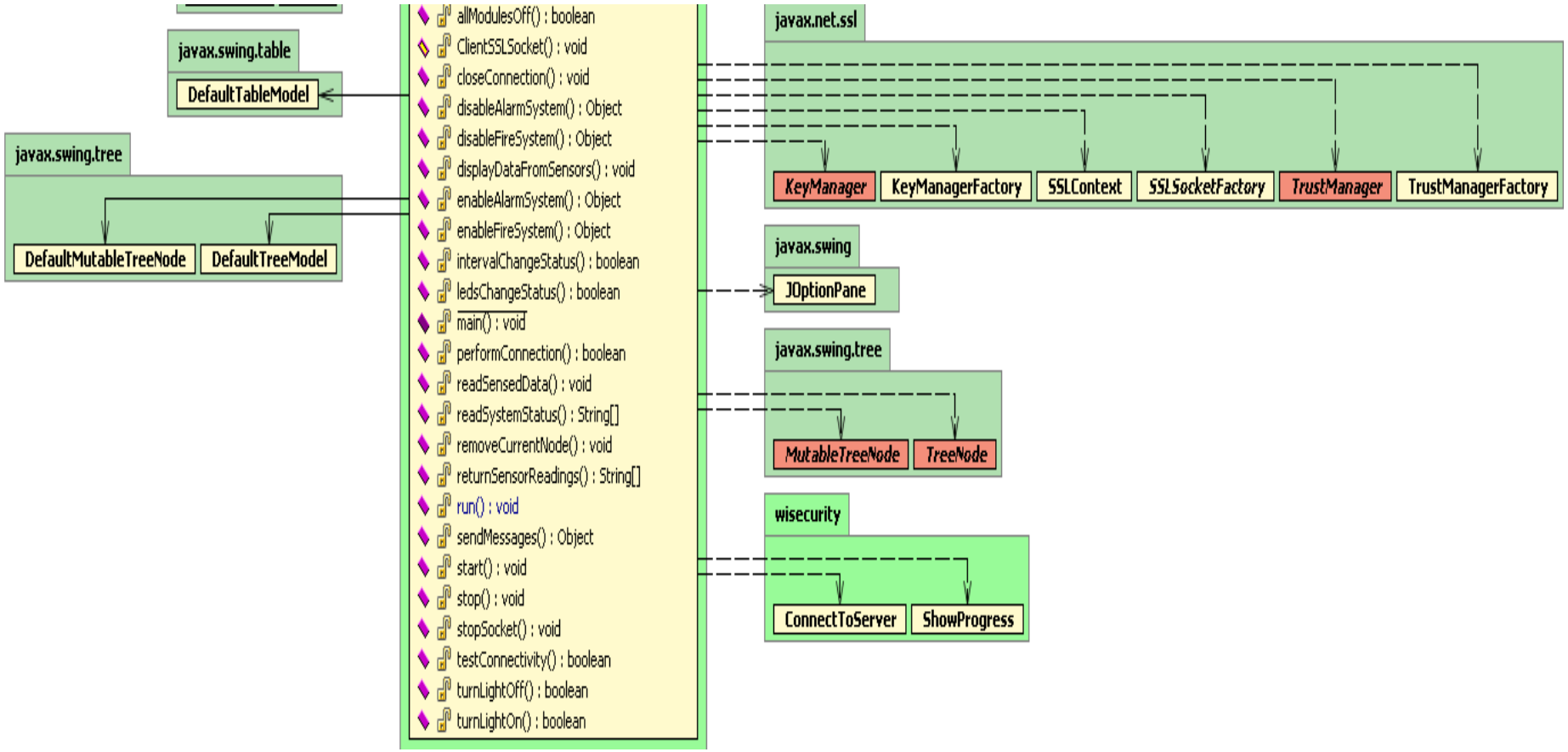


C6. The FirePeriods Class

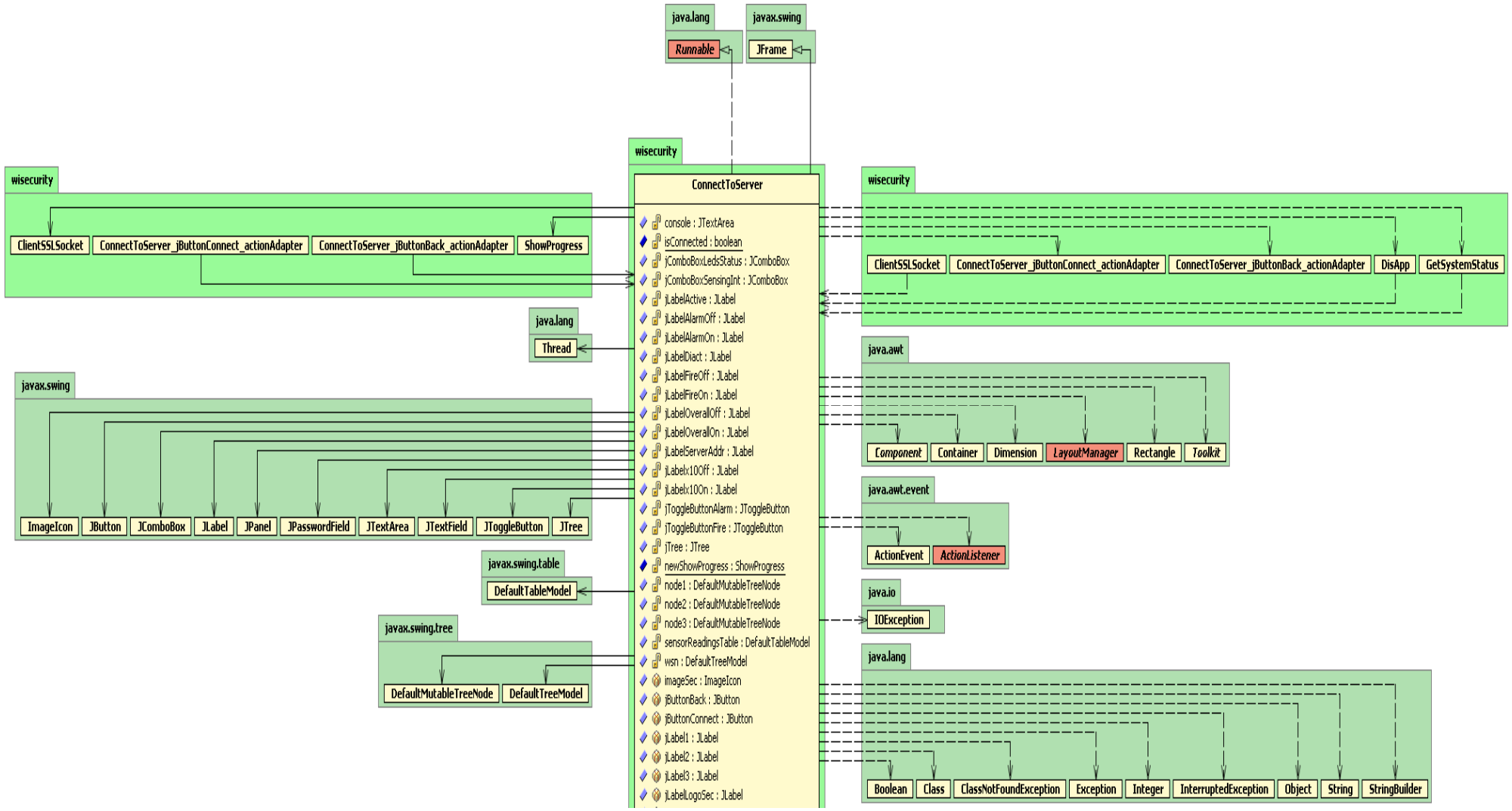


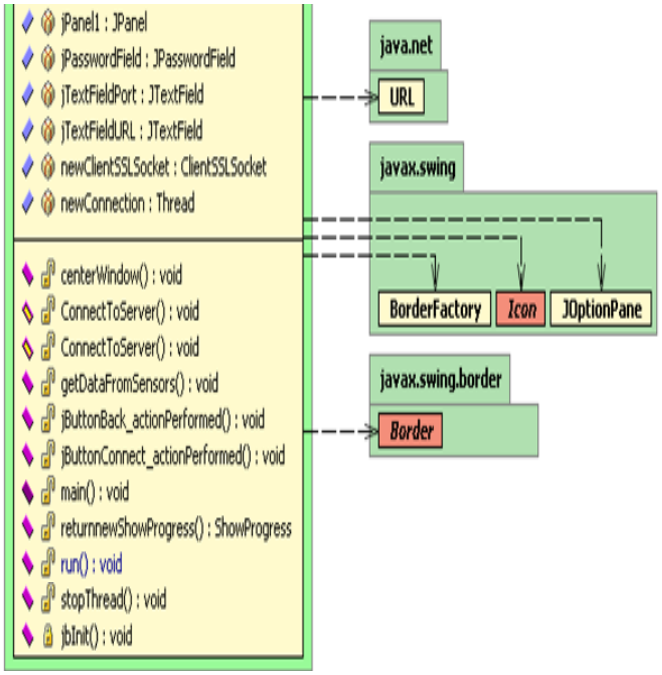
C7. The ClientSSLSocket Class



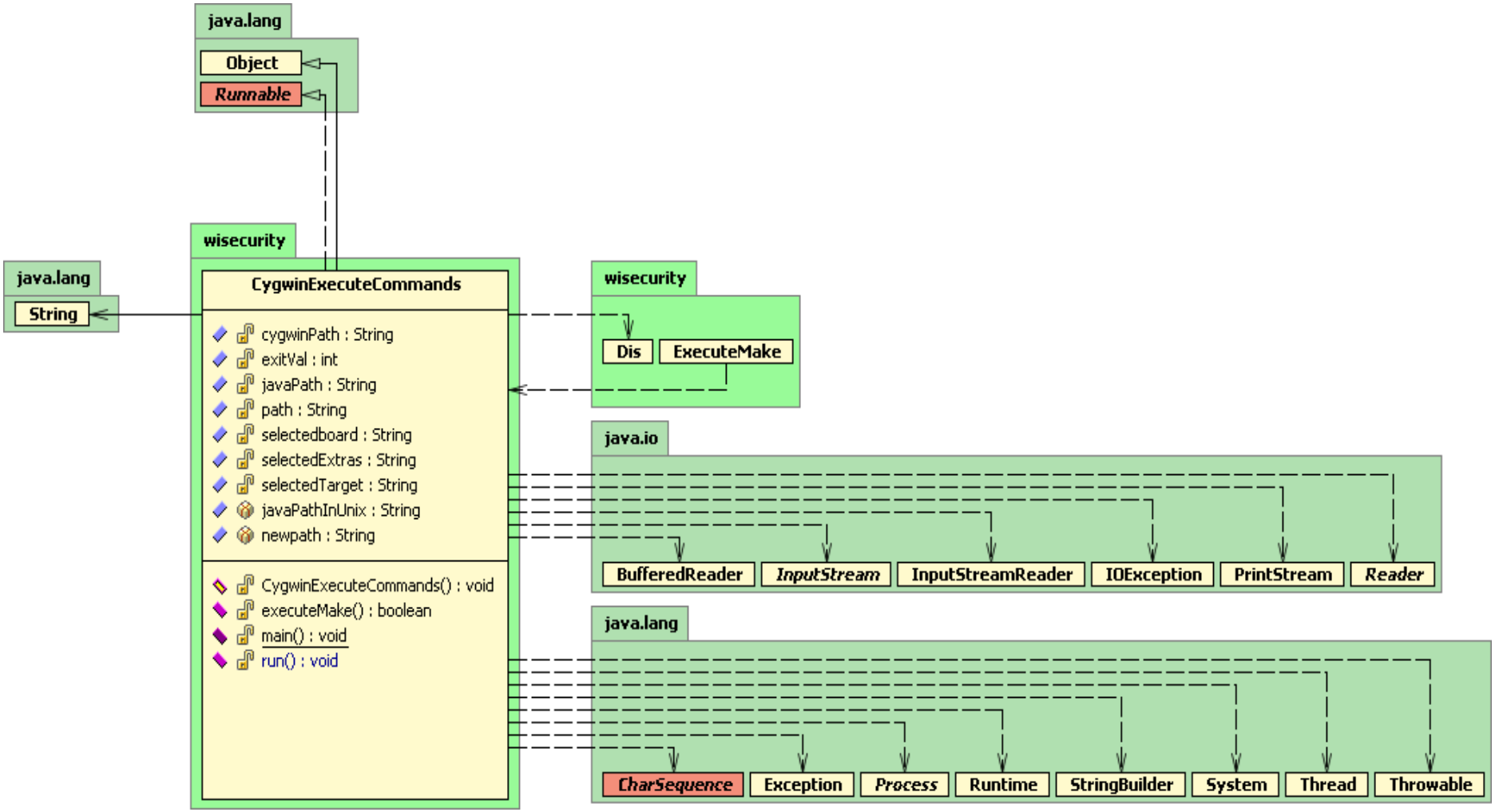

allModulesOff() : boolean
ClientSSLSocket() : void
closeConnection() : void
disableAlarmSystem() : Object
disableFireSystem() : Object
displayDataFromSensors() : void
enableAlarmSystem() : Object
enableFireSystem() : Object
intervalChangeStatus() : boolean
ledsChangeStatus() : boolean
main() : void
performConnection() : boolean
readSensedData() : void
readSystemStatus() : String[]
removeCurrentNode() : void
returnSensorReadings() : String[]
run() : void
sendMessages() : Object
start() : void
stop() : void
stopSocket() : void
testConnectivity() : boolean
turnLightOff() : boolean
turnLightOn() : boolean
javax.swing.table
DefaultTableModel
javax.swing.tree
DefaultMutableTreeNode
DefaultTreeModel
javax.net.ssl
KeyManager
KeyManagerFactory
SSLContext
SSLSocketFactory
TrustManager
TrustManagerFactory
javax.swing
JOptionPane
javax.swing.tree
MutableTreeNode
TreeNode
wisecurity
ConnectToServer
ShowProgress

C8. The ConnectToServer Class

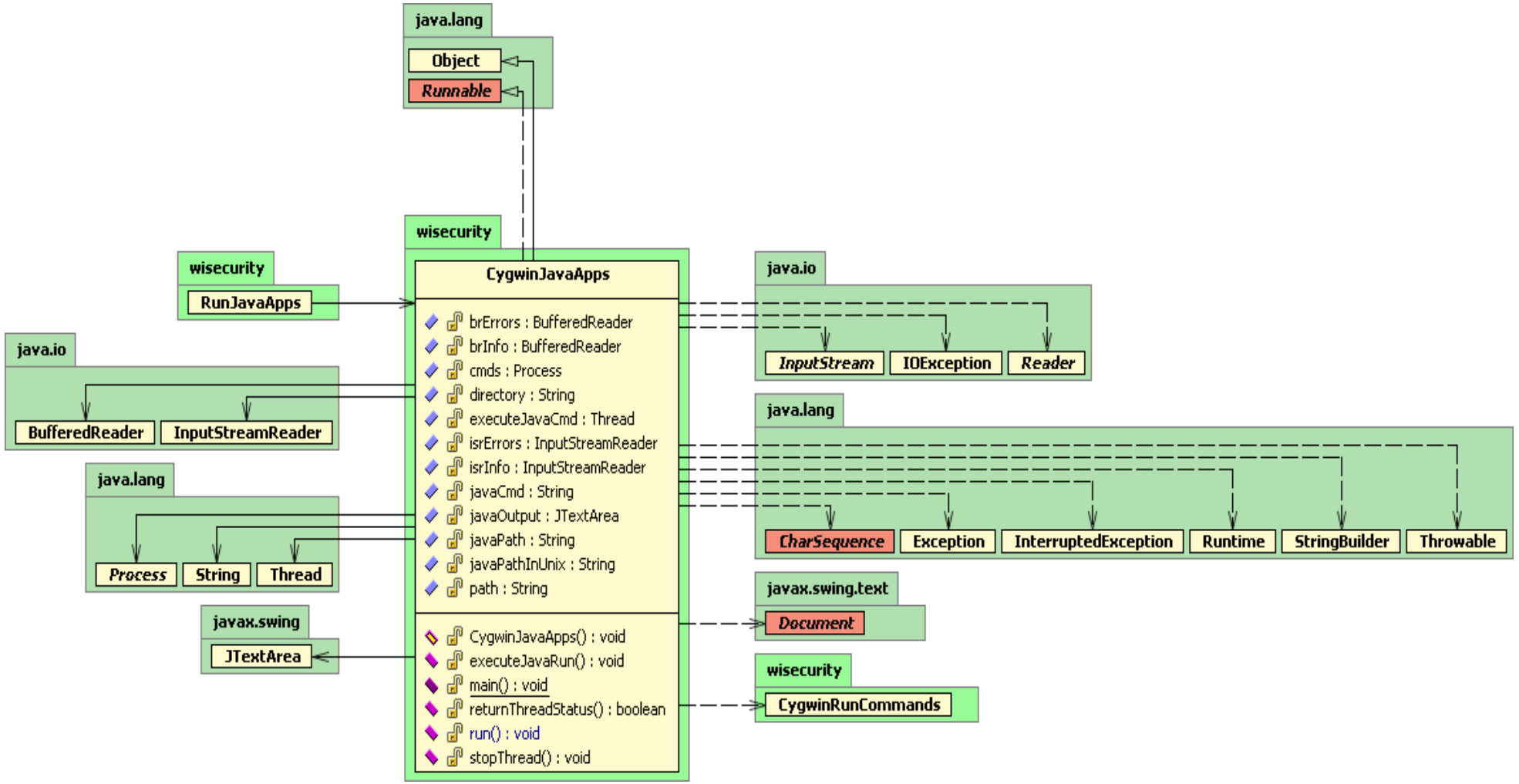



jPanel1 : JPanel
jPasswordField : JPasswordField
jTextFieldPort : JTextField
jTextFieldURL : JTextField
newClientSSLSocket : ClientSSLSocket
newConnection : Thread
centerWindow() : void
ConnectToServer() : void
ConnectToServer() : void
getDataFromSensors() : void
jButtonBack_actionPerformed() : void
jButtonConnect_actionPerformed() : void
main() : void
returnnewShowProgress() : ShowProgress
run() : void
stopThread() : void
jbInit() : void
java.net
URL
javax.swing
BorderFactory
Icon
JOptionPane
javax.swing.border
Border

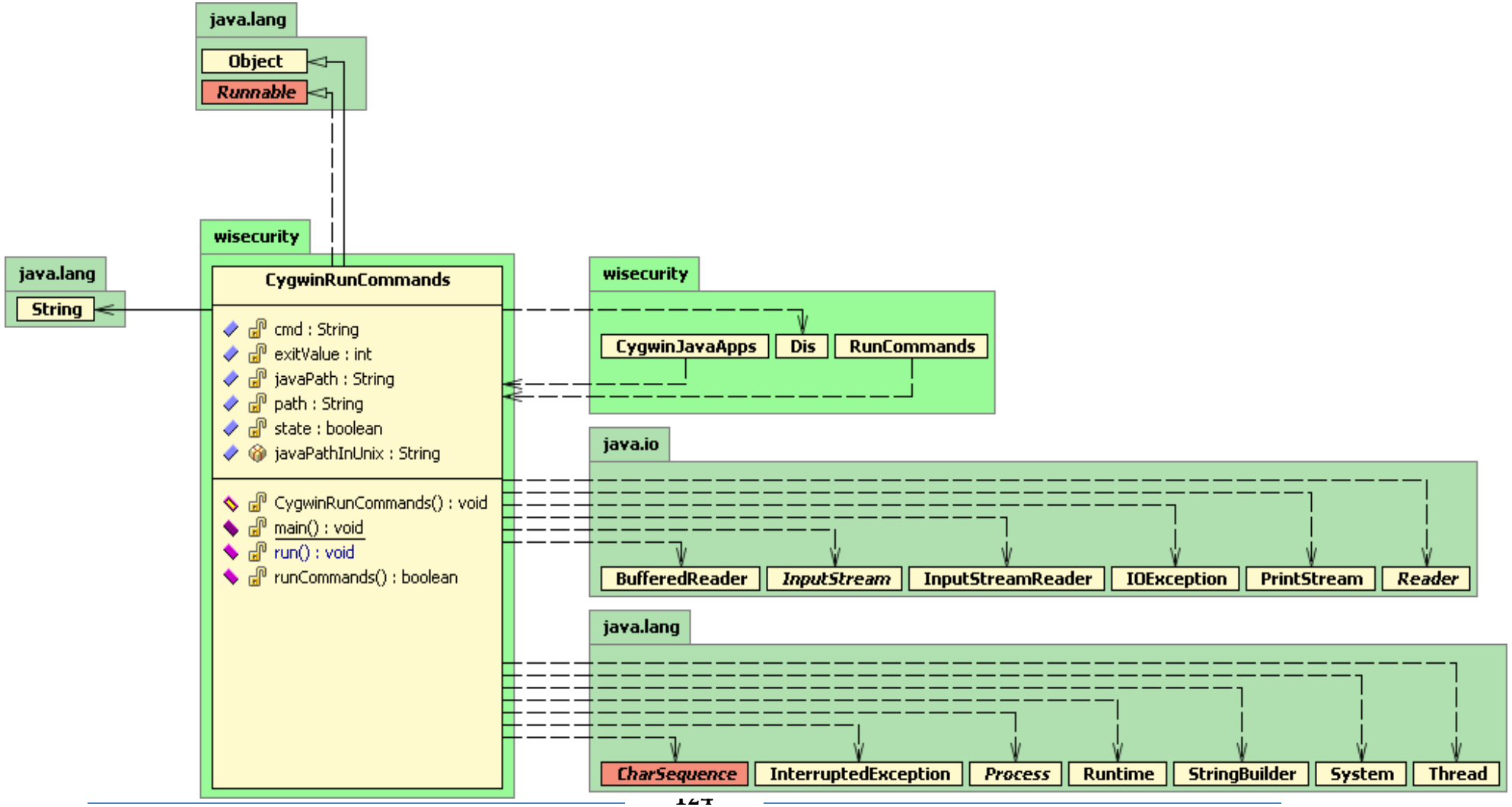
C9. The CygwinExecuteCommands Class



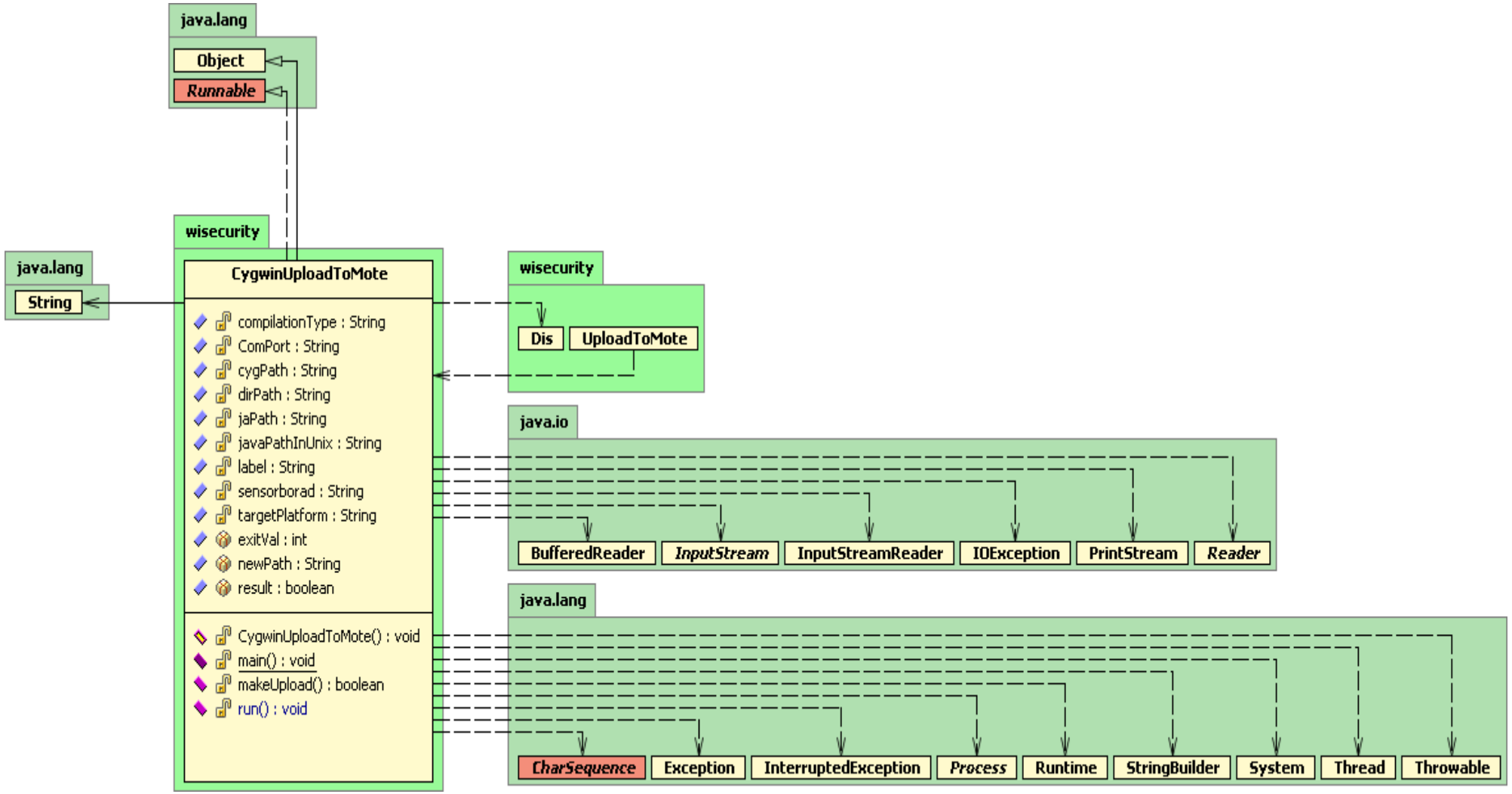
C10. The CygwinJavaApps Class



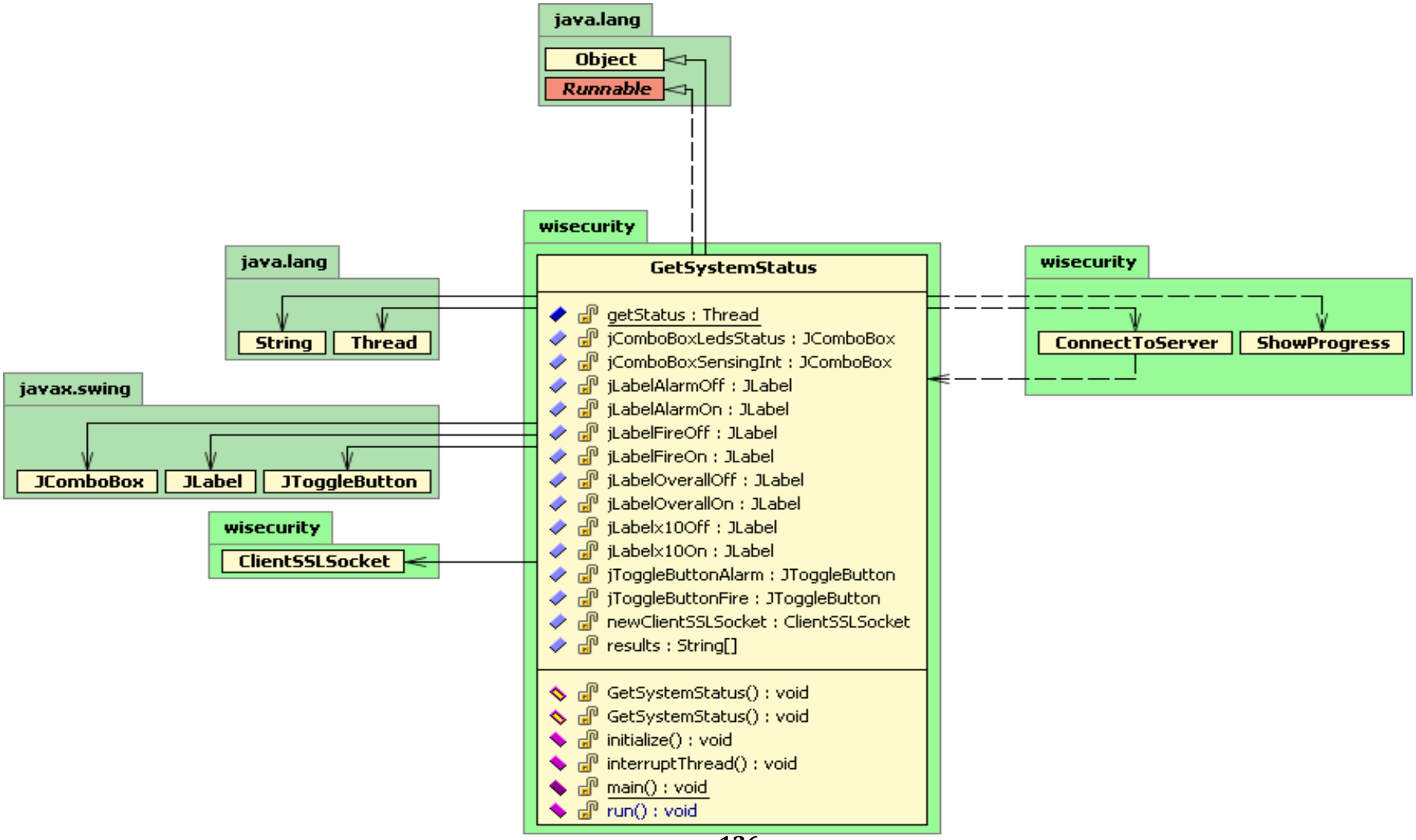
C11. The CygwinRunCommands Class



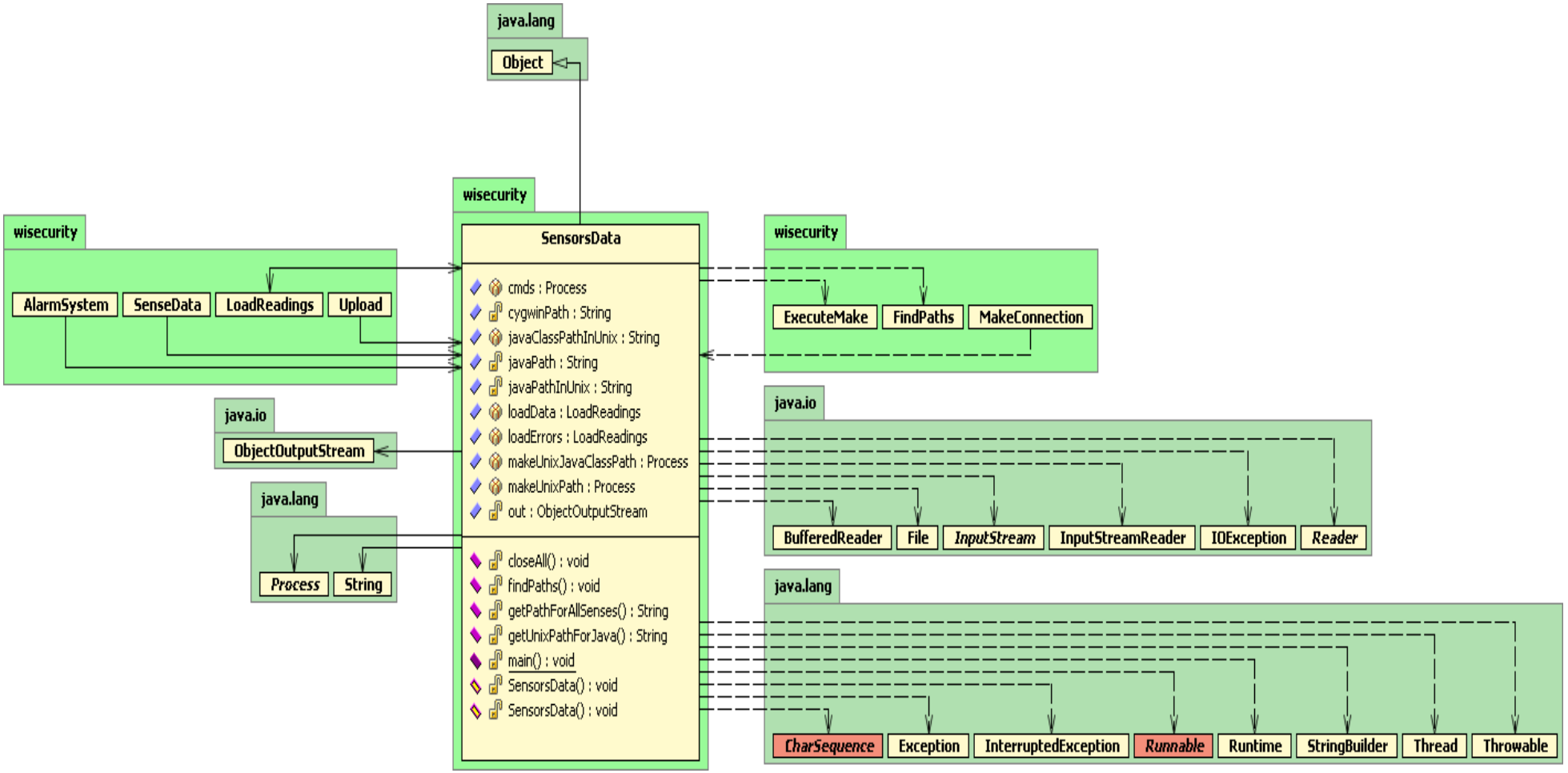
C12. The CygwinUploadToMote Class



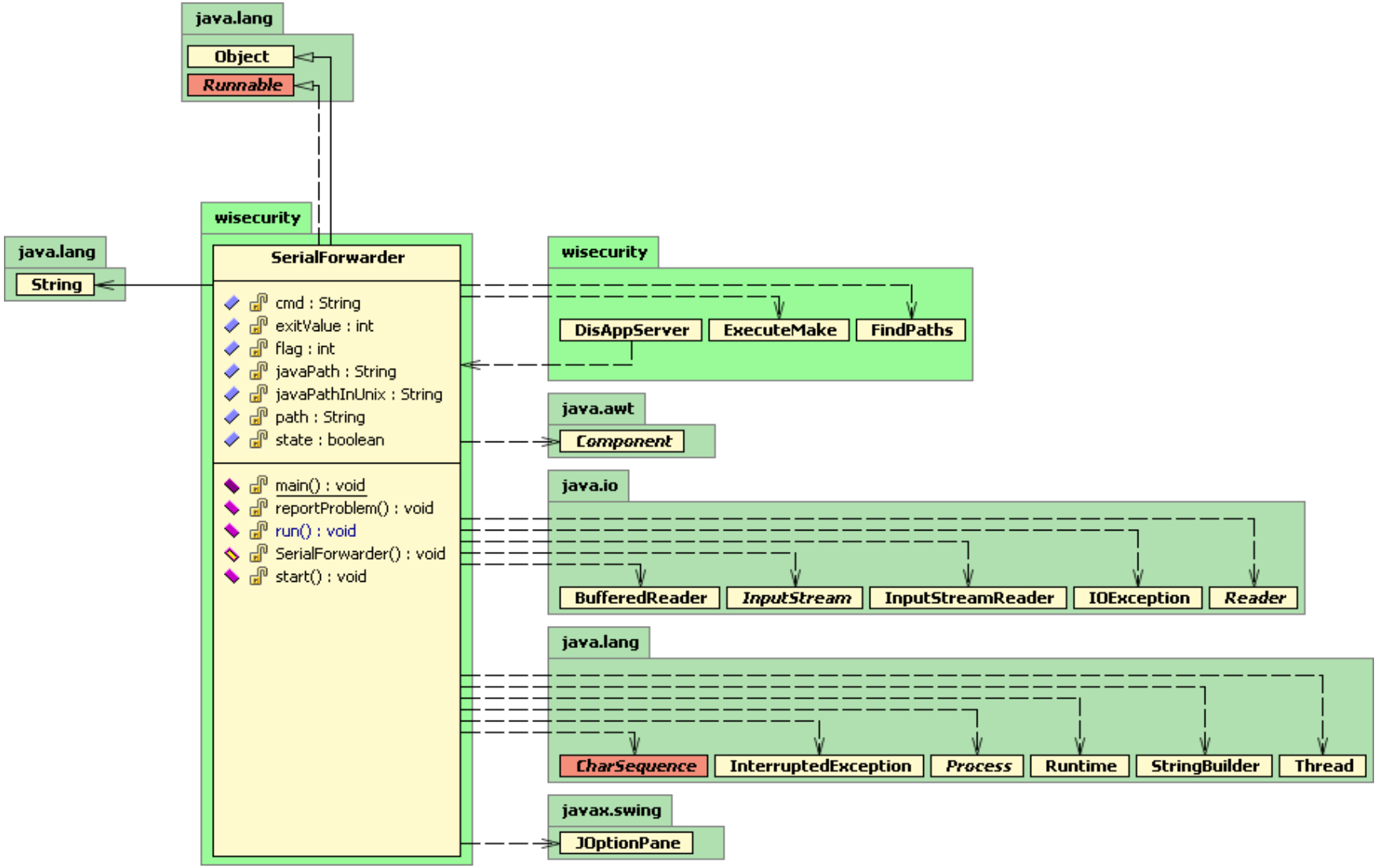
C13. The GetSystemStatus Class



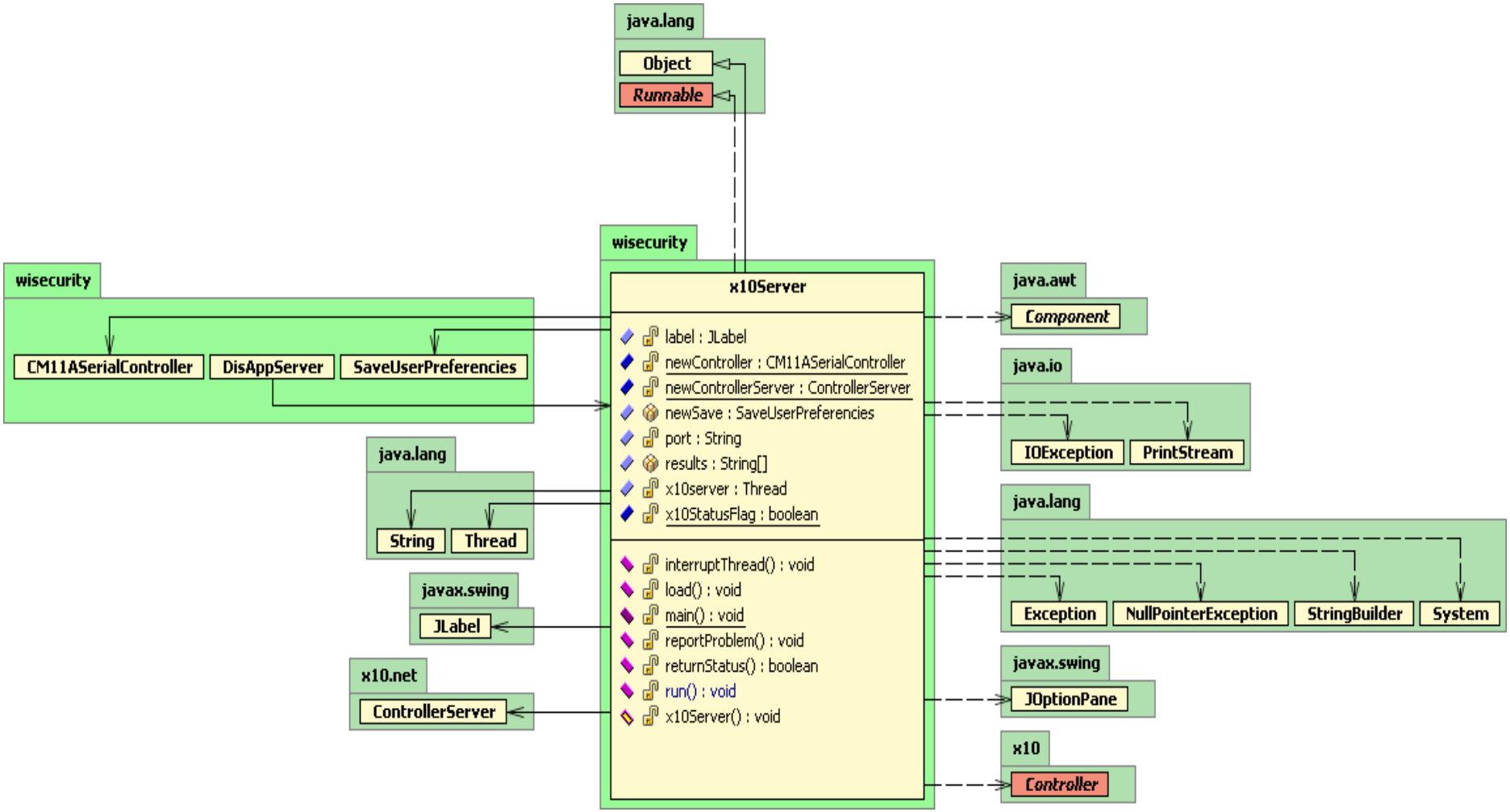
C14. The SensorsData Class



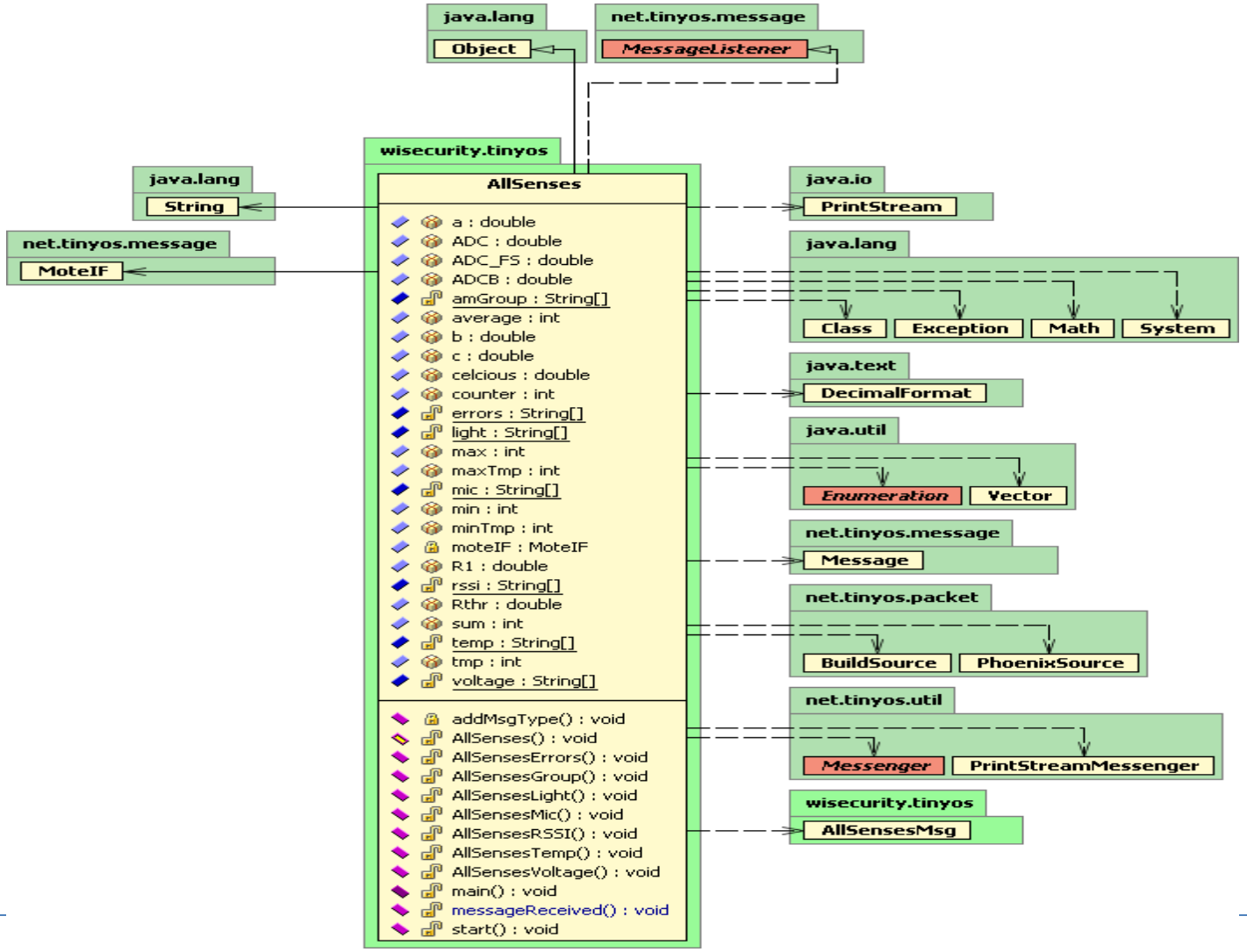
C15. The SerialForwarder Class



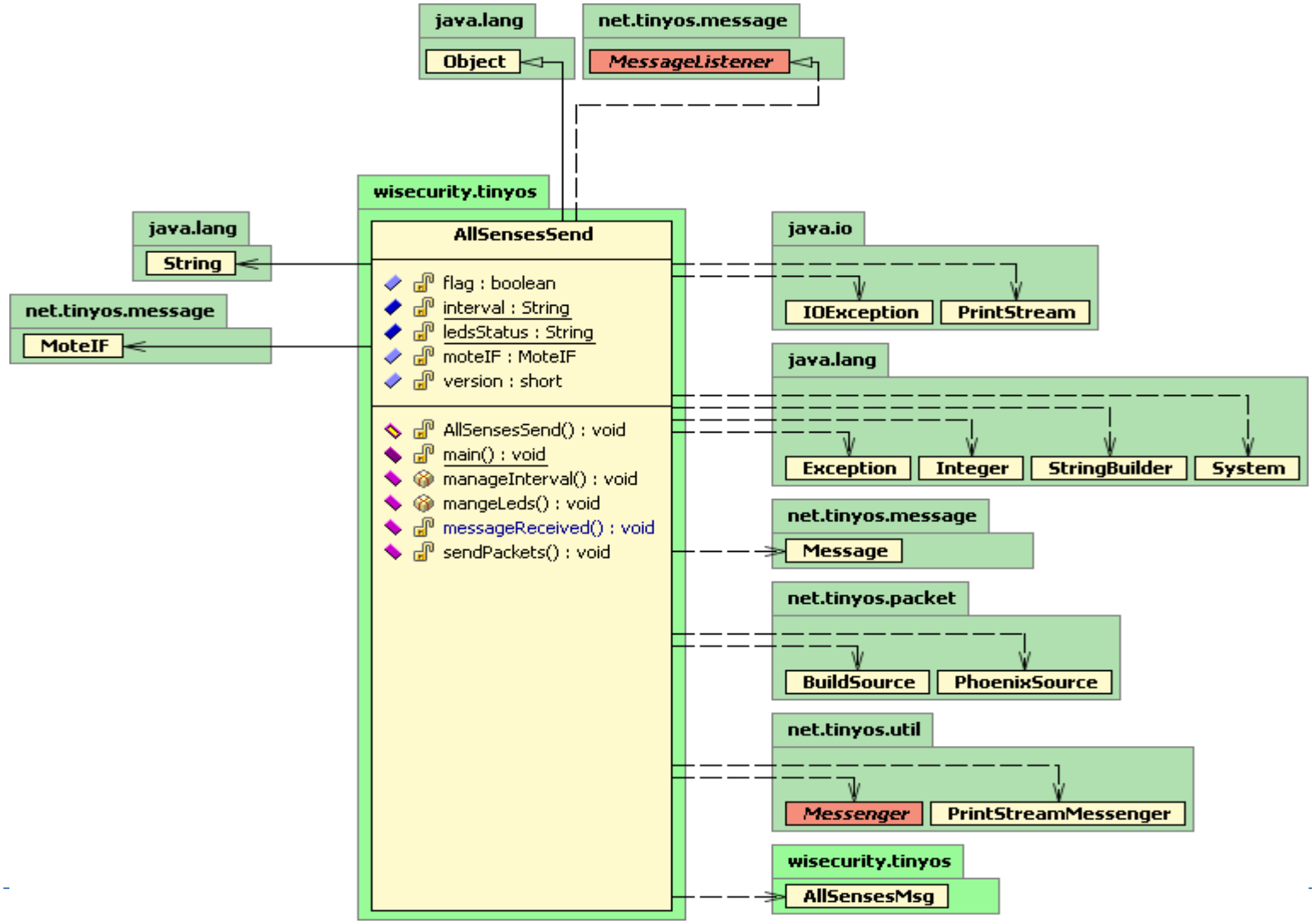
C16. The X10Server Class



C17. The AllSenses Class



C18. The AllSensesSendClass



Appendix D. WiSecurity – Manual

D1. Before Starting

Please read this section carefully, if this is the first time you use the WiSecurity software.

WiSecurity, is a Java implemented software. WiSecurity needs a combination of additional software components, to be previously installed, before running. These software components are used from the WiSecurity on the background during the execution process. Before start WiSecurity you should install the following packages:

- The TinyOS 2.0 distribution,
- The cygwin package,
  - For the TinyOS 2.0 distribution installation instructions , please follow the instructions on the link: http://www.tinyos.net/tinyos-2.x/doc/html/install-tinyos.html
- The JDK or JRE version 1.6 or above.
  - JDK/JRE 1.6 is available from: http://java.sun.com/

Please ensure that both TinyOS and cygwin packages are successfully installed on your system, for not facing with runtime problems. Beyond the above third party software packages you need the following hardware modules:

- An interface board,
- At least two radio boards,
- At least one sensor board,
- A serial (RS - 232) cable or a Serial – to – USB cable,
  - For more information visit: http://www.xbow.com
- An x10 – based Computer Interface module (CM11 or CM11A),
- At least two x10 – based lamp or appliance modules (in order to attach the alarm and the sprinkler device).
  - For more information visit: http://www.intellihome.be

Next, you need to add in your CLASSPATH the TinyOS.jar file path, provided from the TinyOS 2.0 distribution. For adding the TinyOS.jar file path follow the instructions below:

1. Go to Start → Control Panel → System,
2. On the System window, click advanced and then Environment Variables,
3. Go to the System variables and add on the CLASSPATH variable the path of your TinyOS 2.0 jar file, for example:

```
.;C:\your local path\tinyos.jar;
```

Finally, you have to upload the compiled application of AllSenses and BaseStation on each of the nodes of the WSN (Both applications can be found on the CD), starting from the ID: 0 for the base station up to n, where n is the last node of your WSN.

D2. How to use the WiSecurity server

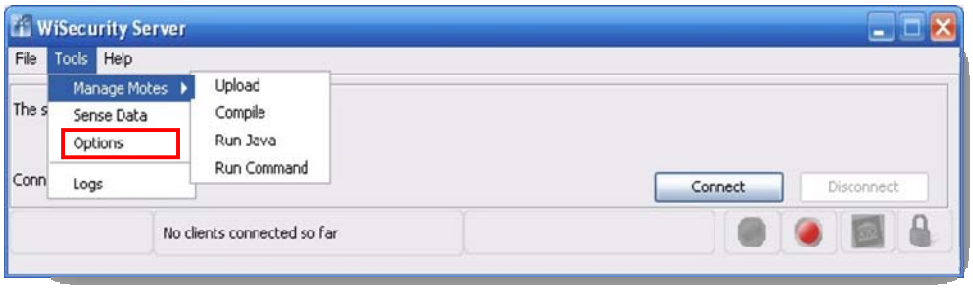
This chapter will guide you on how to configure the WiSecurity server, the x10 network, the security and fire protection systems. Furthermore, you will learn how to load connections history and the system logger.

D2.1. Configure WiSecurity server

Before starting the WiSecurity server, you have to set up a list of properties, including WiSecurity server, x10 network, fire, security and email notification settings.

Step 1: Run the WiSecurity Server,

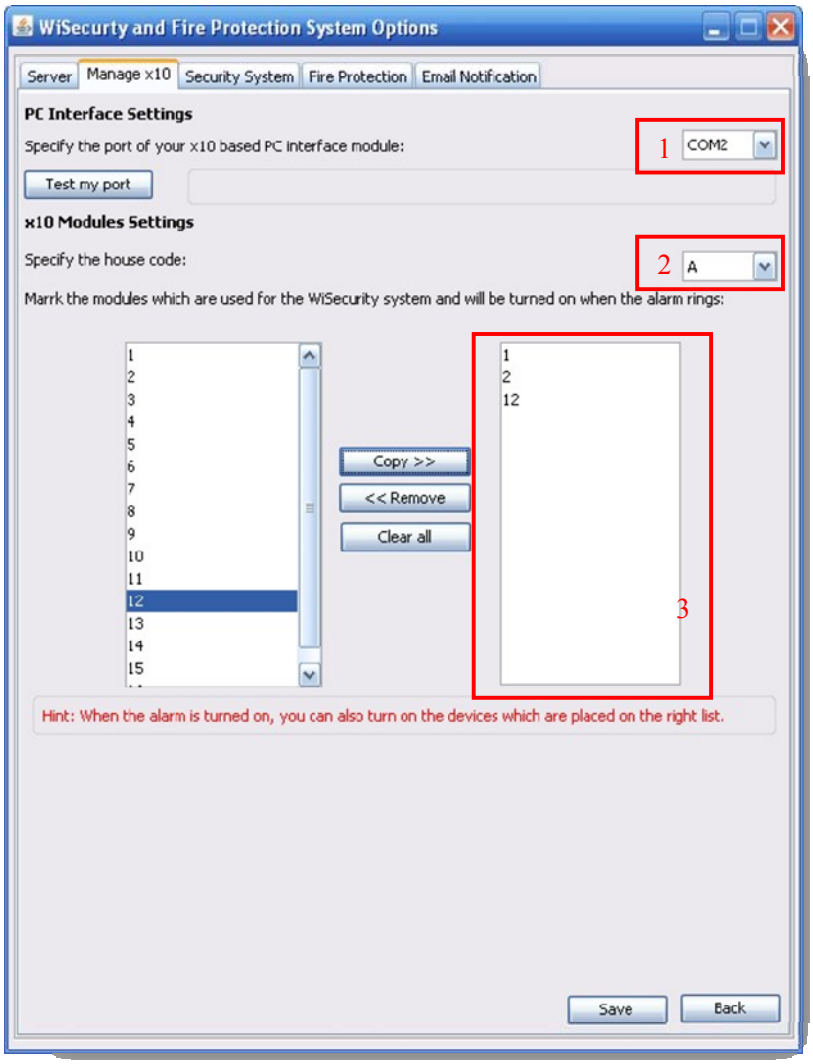
Step 2: Go to Tools → Options.



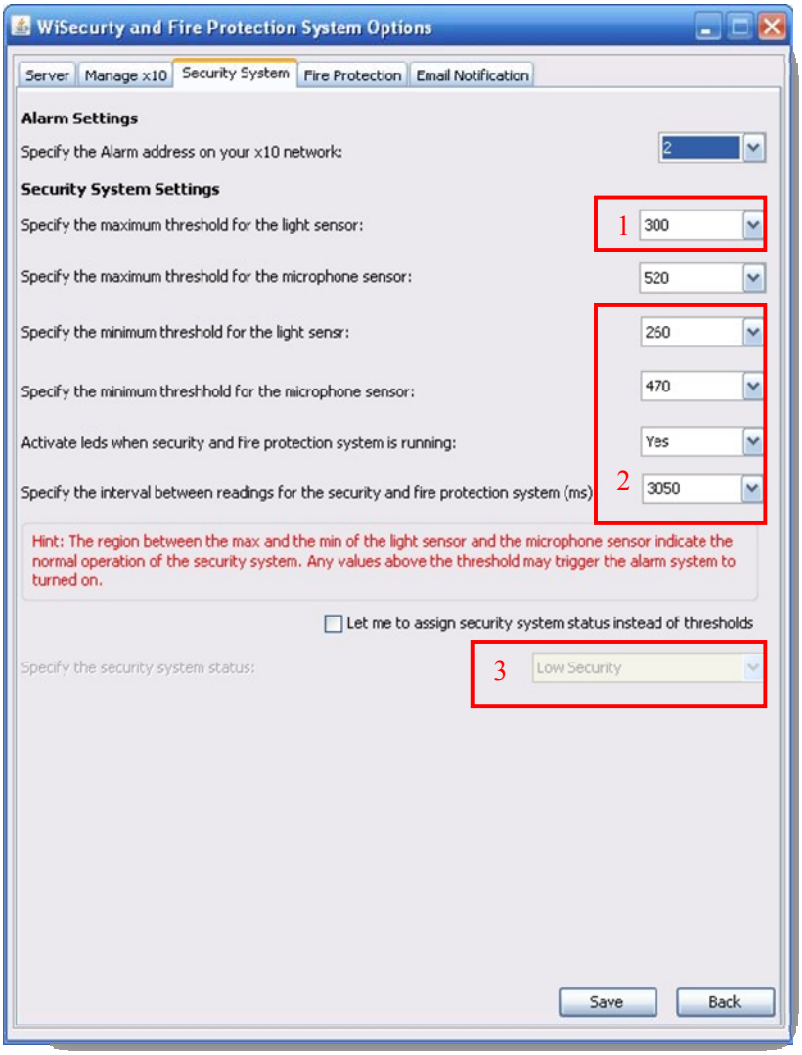
Step 3: Go to the Server tab and set the port number and the concurrent permitted connected clients for the WiSecurity server.

✎ In case you use a firewall on your system, you have to set up your firewall to allow the communication over the selected port.

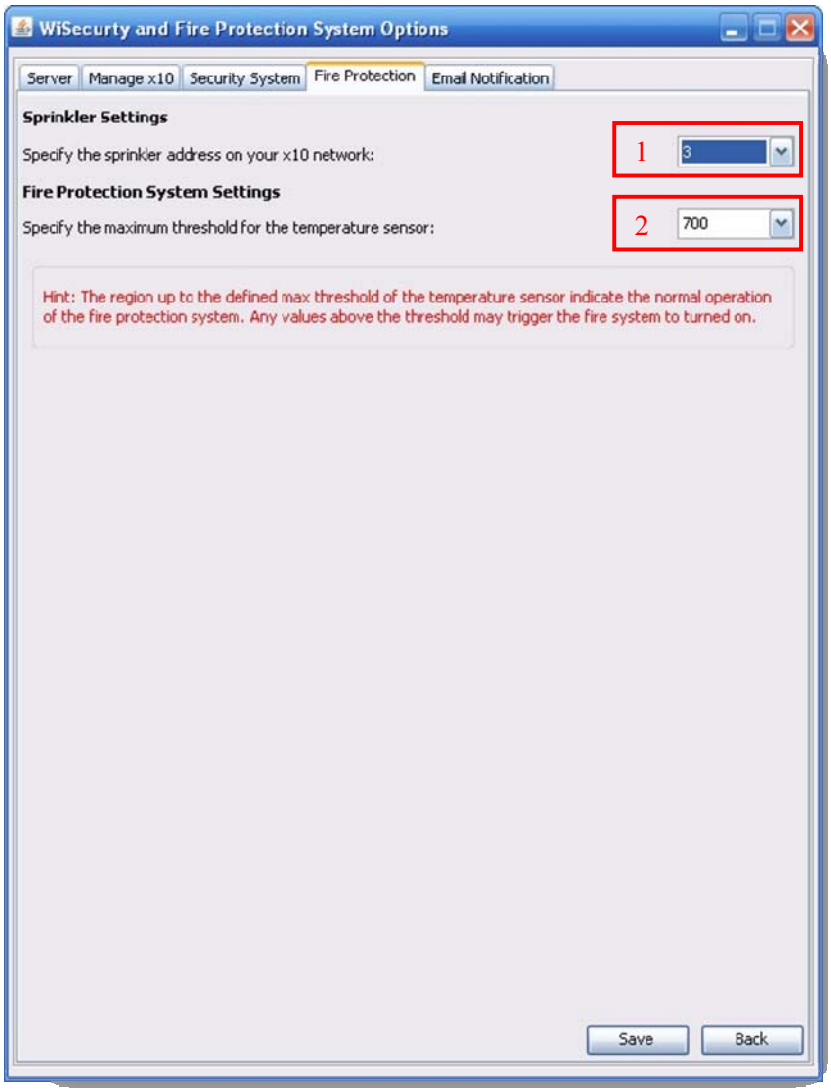
Step 4: Go to the Manage x10 tab. On area one, you can set the communication port of the attached x10 – based interface board. In addition you have to specify, on area two, the house code. This is the address used for the x10 network. It can be from A to P. Finally, on area three, you can add from one to sixteen x10 – based devices (including socket, lamp or appliance modules). These modules will be turned on/ off when the alarm is opened/ closed.



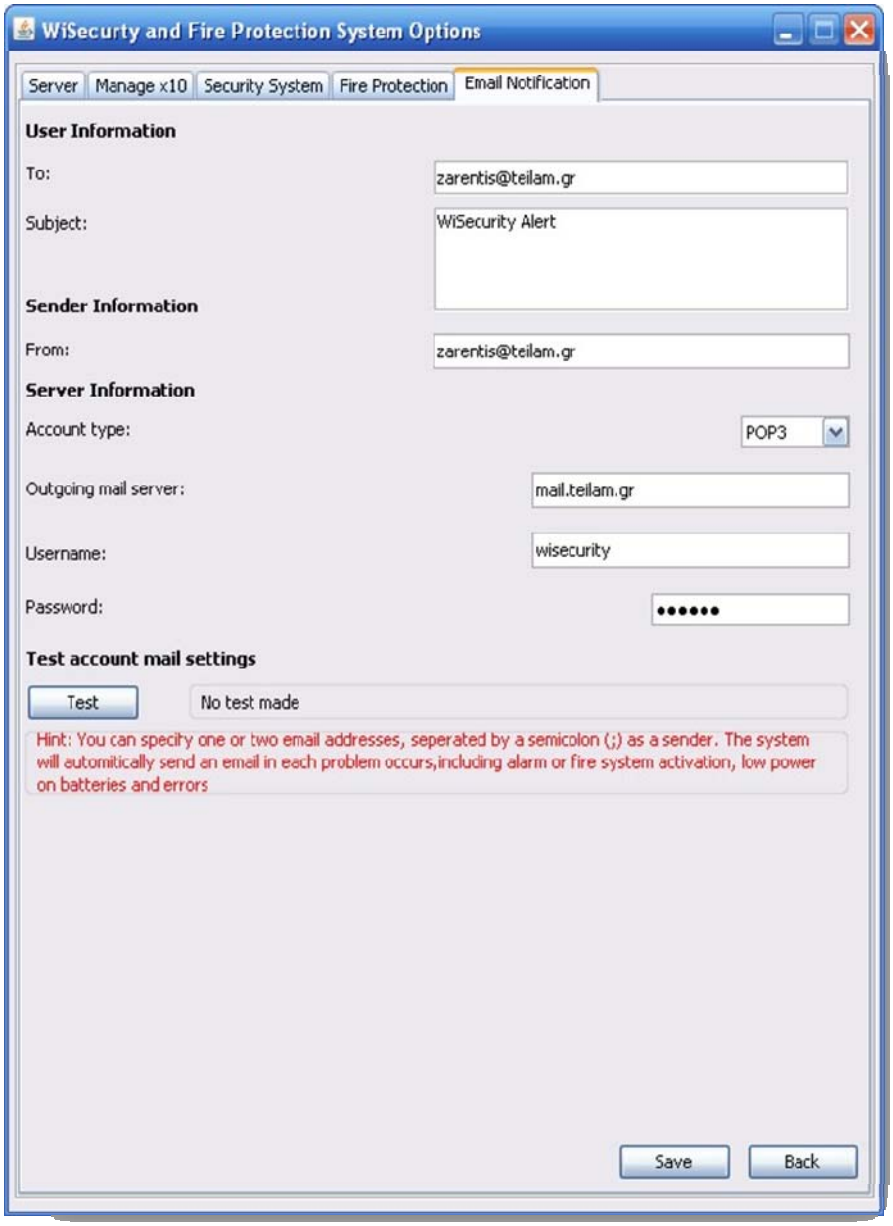
Step 5: Go to the Security System tab. On area one you have to specify the x10 – based alarm device code. This can be a code address from one to sixteen. On area two you can assign the minimum and the maximum values for the light and microphones sensors. Values which are between the assigned maximum and minimum are concerned as normal operation values for the protection system, while values which are less or greater to the minimum or maximum assigned values are concerned as non – normal operation values and may cause the alarm system to turn on. Next, on area three you can set the leds status and the sensing interval for the sensor devices on the WSN. Finally, in case you are not familiar with raw data coming from the WSN you can choose on area four, among the three available security levels. These are high, normal or low security level.


WiSecurty and Fire Protection System Options
Server
Manage x10
Security System
Fire Protection
Email Notification
Alarm Settings
Specify the Alarm address on your x10 network:
2
Security System Settings
Specify the maximum threshold for the light sensor:
1
300
Specify the maximum threshold for the microphone sensor:
520
Specify the minimum threshold for the light sensr:
250
Specify the minimum threshhold for the microphone sensor:
470
Activate leds when security and fire protection system is running:
Yes
Specify the interval between readings for the security and fire protection system (ms)
2
3050
Hint: The region between the max and the min of the light sensor and the microphone sensor indicate the normal operation of the security system. Any values above the threshold may trigger the alarm system to turned on.
Let me to assign security system status instead of thresholds
Specify the security system status:
3
Low Security
Save
Back

Step 6: Go to the Fire Protection tab. On area one, you have to specify the x10 – based sprinkler device code. On area two, you need to specify the maximum normal limit for temperature. Any value above this limit may cause the sprinkler device to turn on.

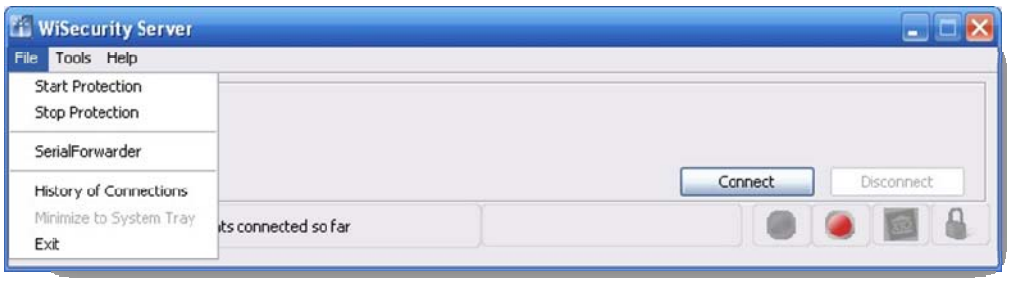

WiSecurty and Fire Protection System Options
Server
Manage x10
Security System
Fire Protection
Email Notification
Sprinkler Settings
Specify the sprinkler address on your x10 network:
1
3
Fire Protection System Settings
Specify the maximum threshold for the temperature sensor:
2
700
Hint: The region up to the defined max threshold of the temperature sensor indicate the normal operation of the fire protection system. Any values above the threshold may trigger the fire system to turned on.
Save
Back

Step 7: (Optional) Go to the Email Notification tab. Here you can add your email server settings for receiving email notifications each time the alarm or sprinkler device is turned on or off.


WiSecurty and Fire Protection System Options
Server
Manage x10
Security System
Fire Protection
Email Notification
User Information
To:
zarentis@teilam.gr
Subject:
WiSecurity Alert
Sender Information
From:
zarentis@teilam.gr
Server Information
Account type:
POP3
Outgoing mail server:
mail.teilam.gr
Username:
wisecurity
Password:
Test account mail settings
Test
No test made
Hint: You can specify one or two email addresses, seperated by a semicolon (;) as a sender. The system will automitically send an email in each problem occurs,including alarm or fire system activation, low power on batteries and errors
Save
Back

Before enabling WiSecurity server, you have to set the server settings. If you skip section D2.1, go back and follow the instructions.

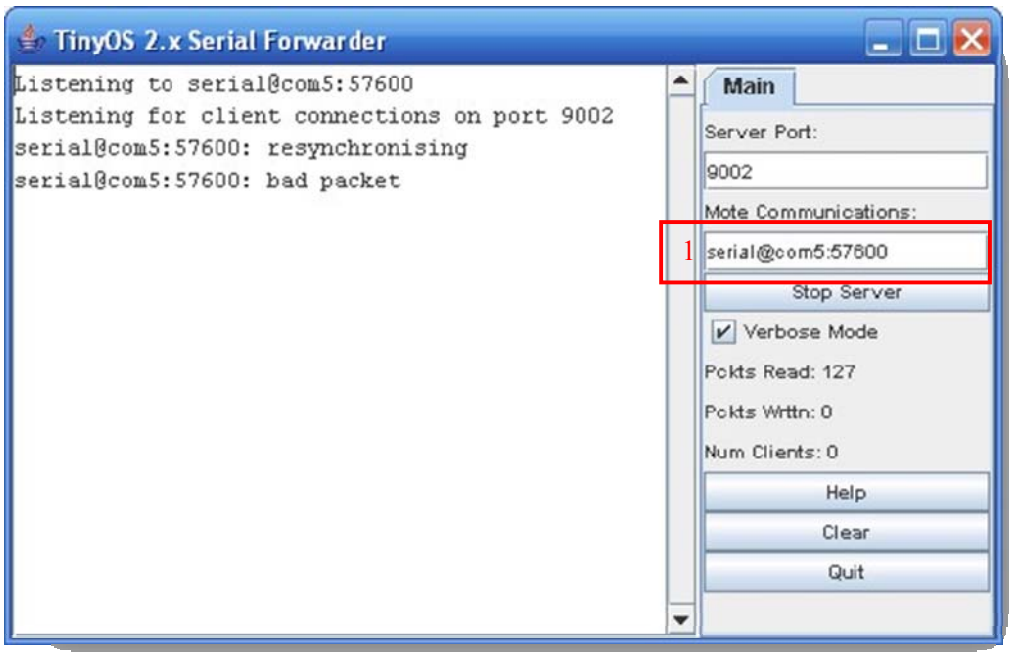
Before starting the WiSecurity server, you have to start the SerialForwarder application. SeialForwarder, acts as a server for the WSN. It gathers data from the WSN and forwards that data to the WiSecurity server. For running the SerialForwarder application, go to:



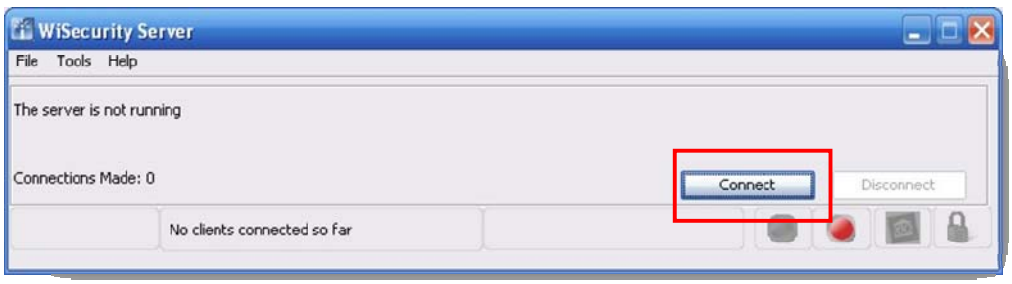
File → SerialForwarder

You should see something like the picture below. On area one, you should set the communication port where your interface board is attached. The default port is the communication port 1 (com1). In addition you have to specify the radio board which yours WSN use and the baud rate. Finally, press start server. If your WSN is set correctly, the packets read counter should increase, each time a new packet arrives on the interface board from the network.

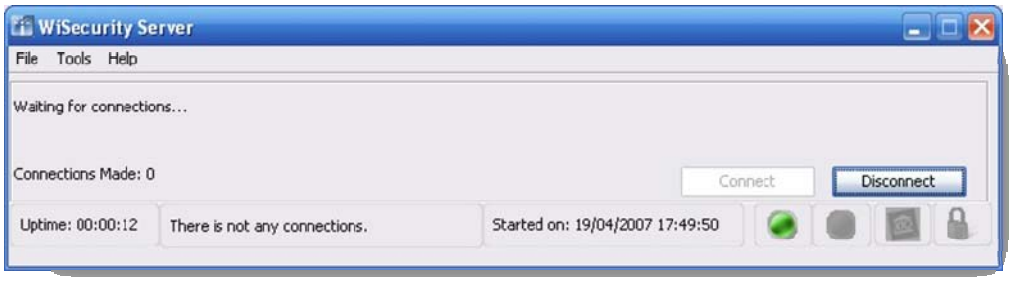
> ✎ For more information regarding the port, the radio board and the baud rate, refer to the TinyOS 2.0 online tutorials.



Next, minimize the SerialForwarder application and maximize the WiSecurity server application. Press the Connect button for initializing the WiSecurity server.



If you set up correctly the server you should see the picture below. The green colored image, on the right of the application indicates that the server is successfully running.

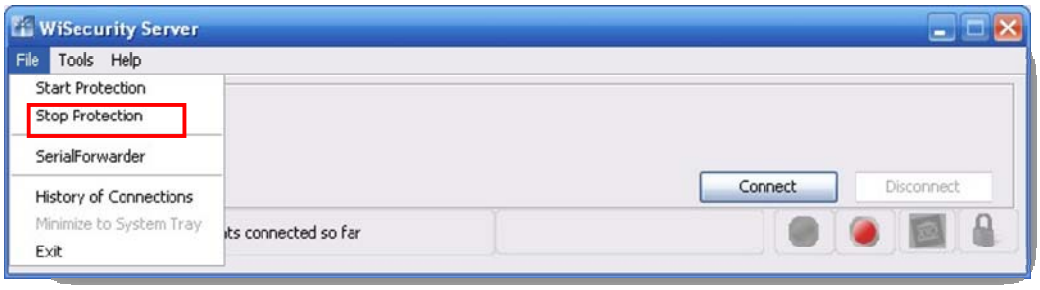


At any time, you can close the WiSecurity server, by pressing the disconnect button.

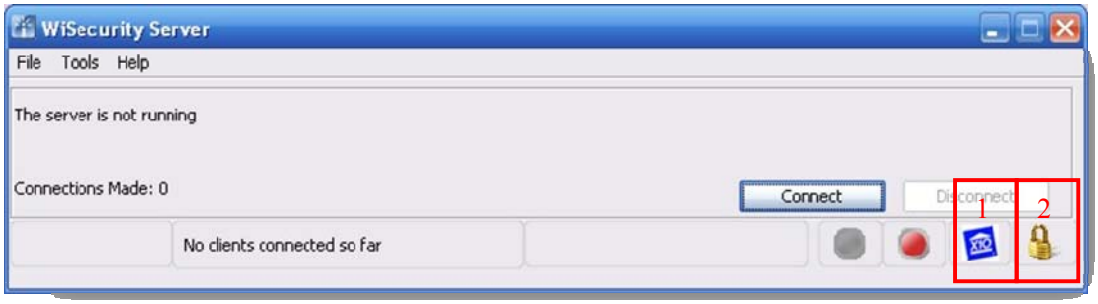
D2.3. Running the Security and Fire Protection systems

Before enabling the Security and Fire Protection system, you have to set the x10 network, security and fire protection settings. If you skip section D2.1, go back and follow the instructions.

Security system manages the alarm device while fire system manages the sprinkler device. Both systems operation depends on the options you assign. For enabling the Security and Fire protections systems go to File → Start Protection.



If you successfully, install the TinyOS 2.0 distribution and the cygwin package you should see the picture below. The icon on area one, indicates that the x10 network is successfully running, and the icon on area two indicates that the security and fire protection systems are properly initialized.



In case that, icon on area one remains disabled means that the WiSecurity is not able to communicate with the x10 – based Computer Interface module. If icon, on area two remains disabled means that:

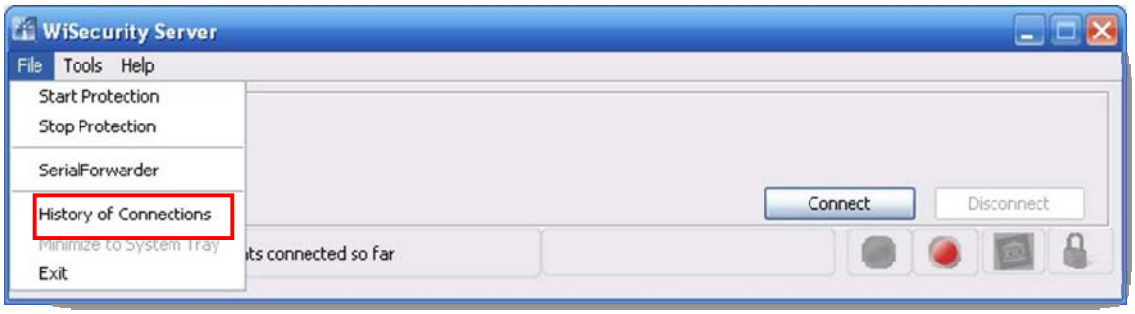
- WiSecurity, could not load data from the WSN,
- WiSecurity, could not find the interface board,
- The TinyOS 2.0 distribution is not running properly or not installed correctly,
- The cygwin distribution is not running properly or not installed correctly,
- You have not installed JDK/JRE version 1.6 or above or your system`s primary JDK/JRE version is not the version 1.6.

At any time, you can close the security and fire protection system, by choosing File → Stop Protection.

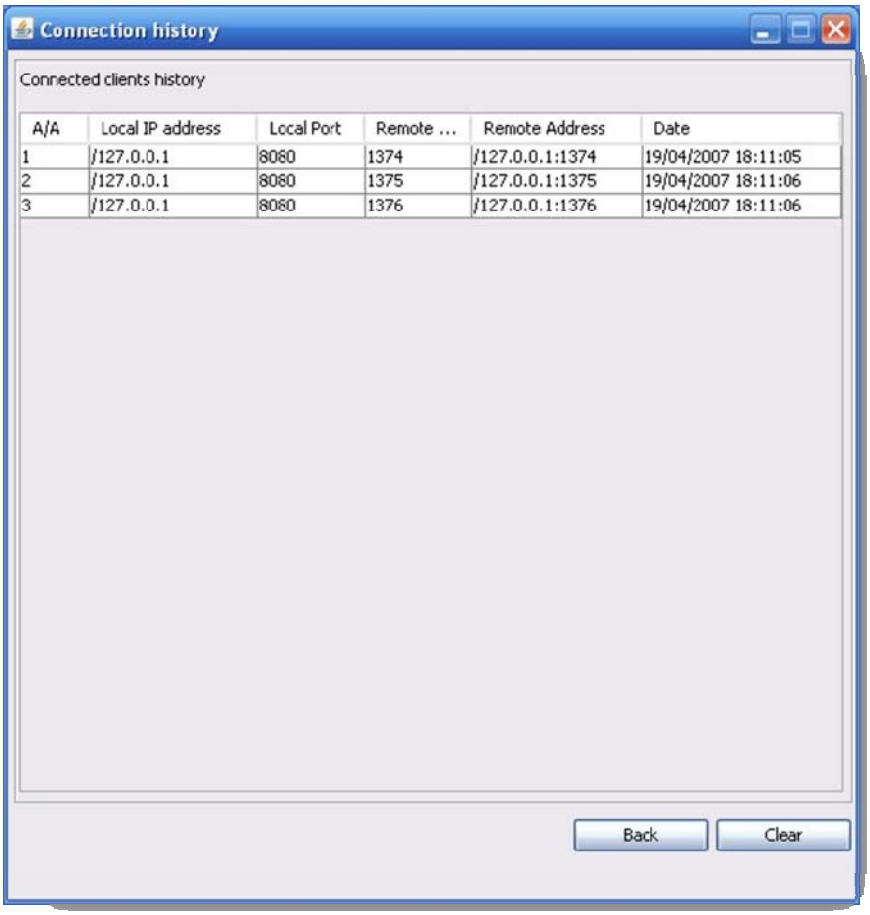
D2.4. Load Connections History

WiSecurity holds the last fifty connected clients with the server. In order to load the connections history, go to:

File → History of Connections



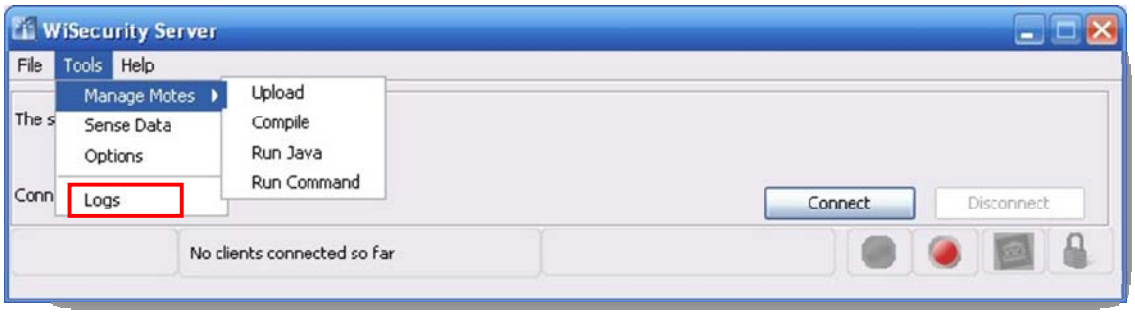
If one or more clients have already connected with the WiSecurity server, you should see something like the picture below.



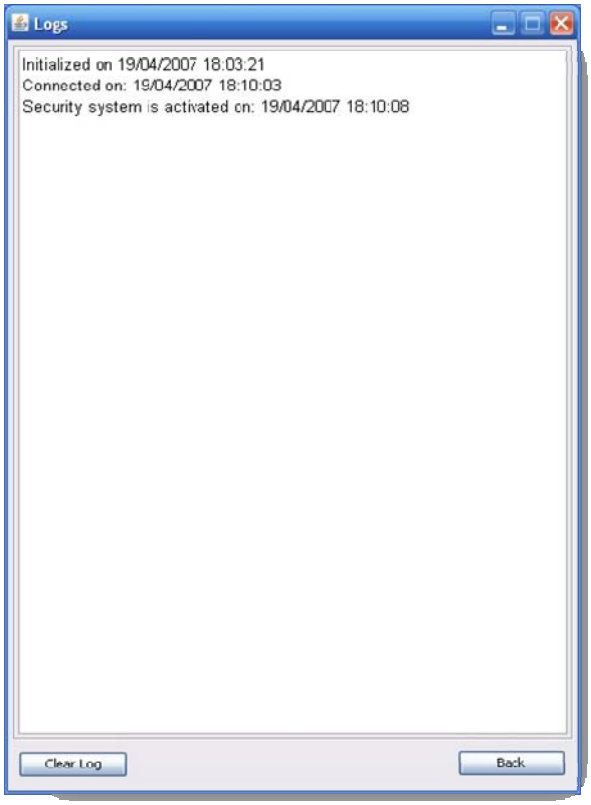
D2.5. Load the WiSecurity Logger

WiSecurity holds a logger regarding WiSecurity operation functions or execution errors. In order to see the logger, go to:

Tools → Logs



You should see something like the picture below.

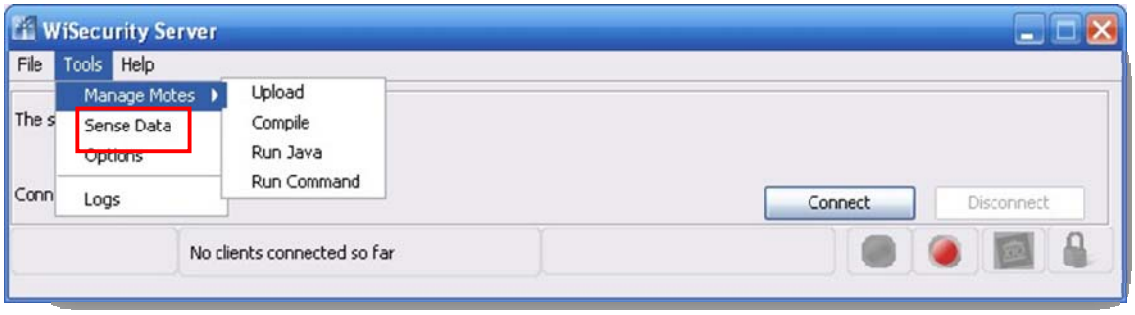


D2.6. Collect WSN Data

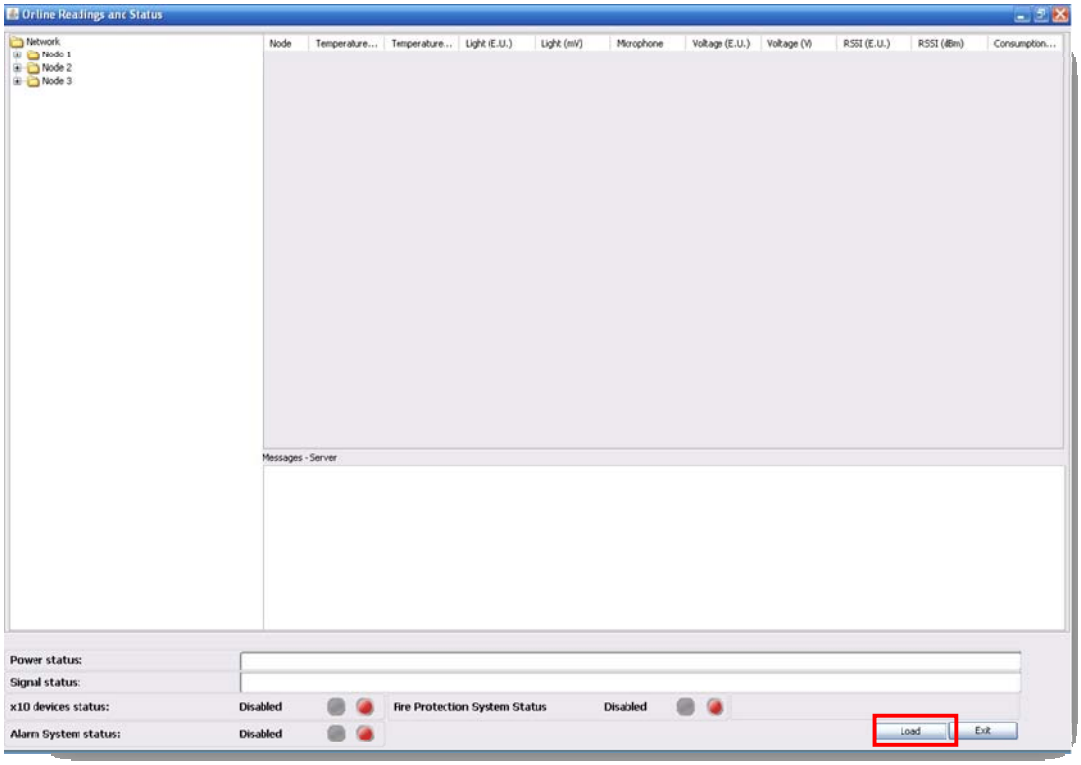
WiSecurity includes the capability of reading data from the network and displaying it on screen. In addition you are able to check the status of the x10 network, the alarm

protection system and the fire protection system using a collection of easy – to – read icons. For running the WSN data collector go to:

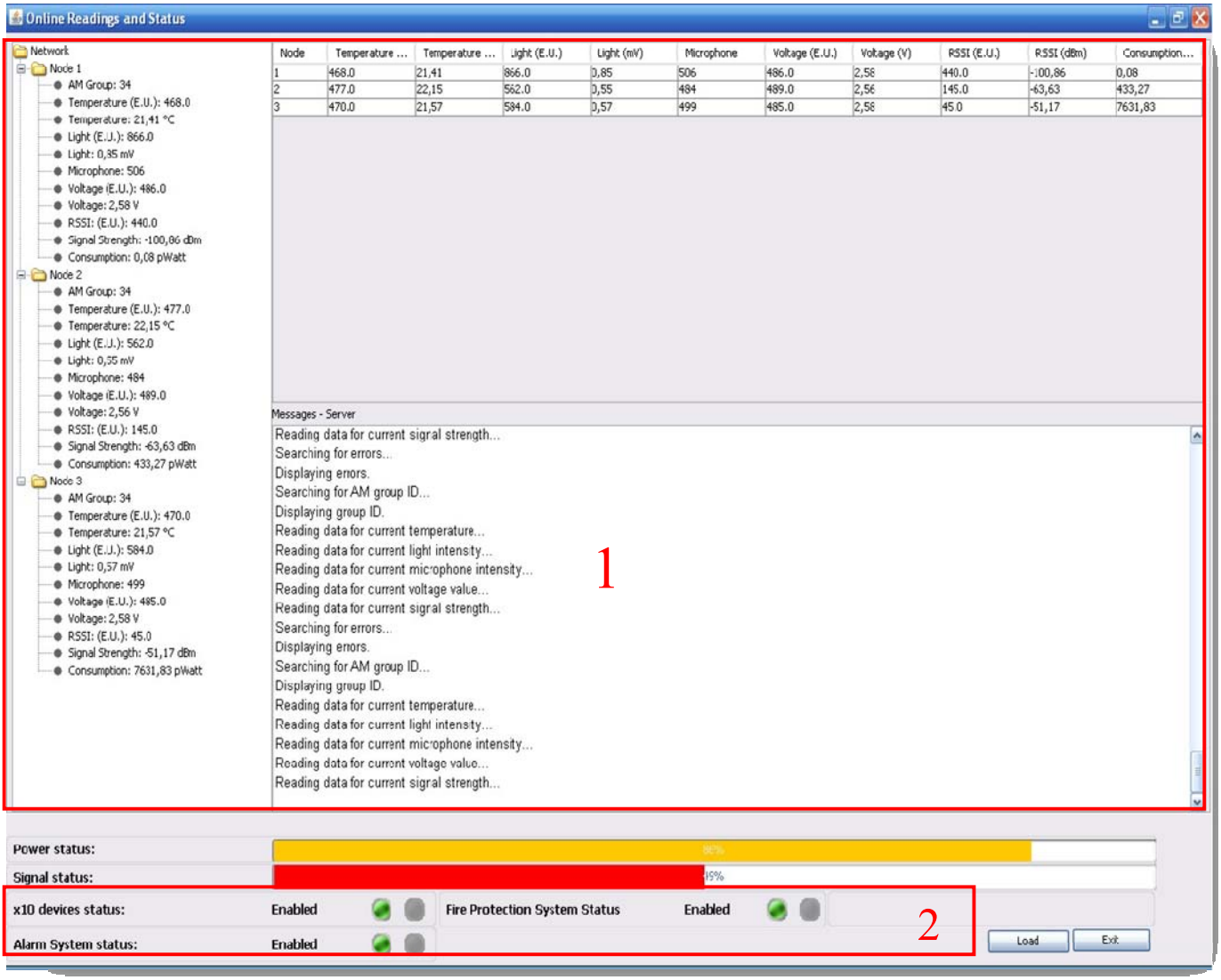
Tools → Sense Data



A new window will appear like the picture below. Press the load button, for reading data from the WSN.



WiSecurity, will try to connect to the SerialForwarder application (if SerialForwarder is not running, start it now), gather data from the nodes and display it on area one.



Additionally, on area two you can see the current status of the x10 network, the alarm protection and the fire protection systems.

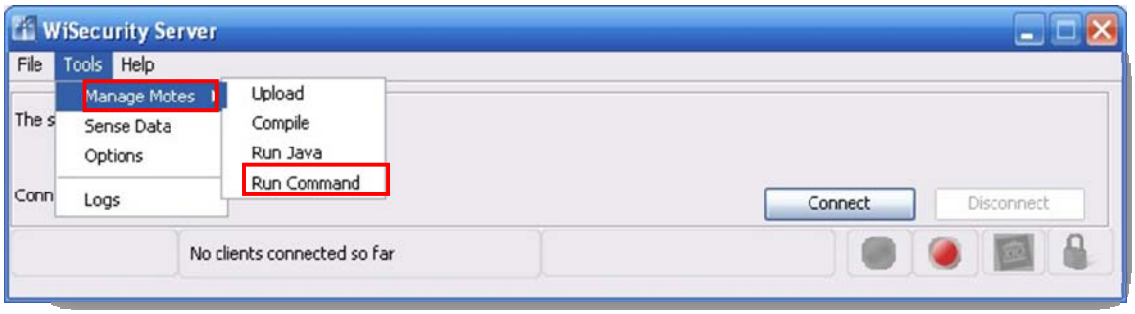
D3. How to manage nodes

This chapter will guide you on how to upload new applications on your nodes, compile new applications and run any cygwin compatible command using WiSecurity environment.

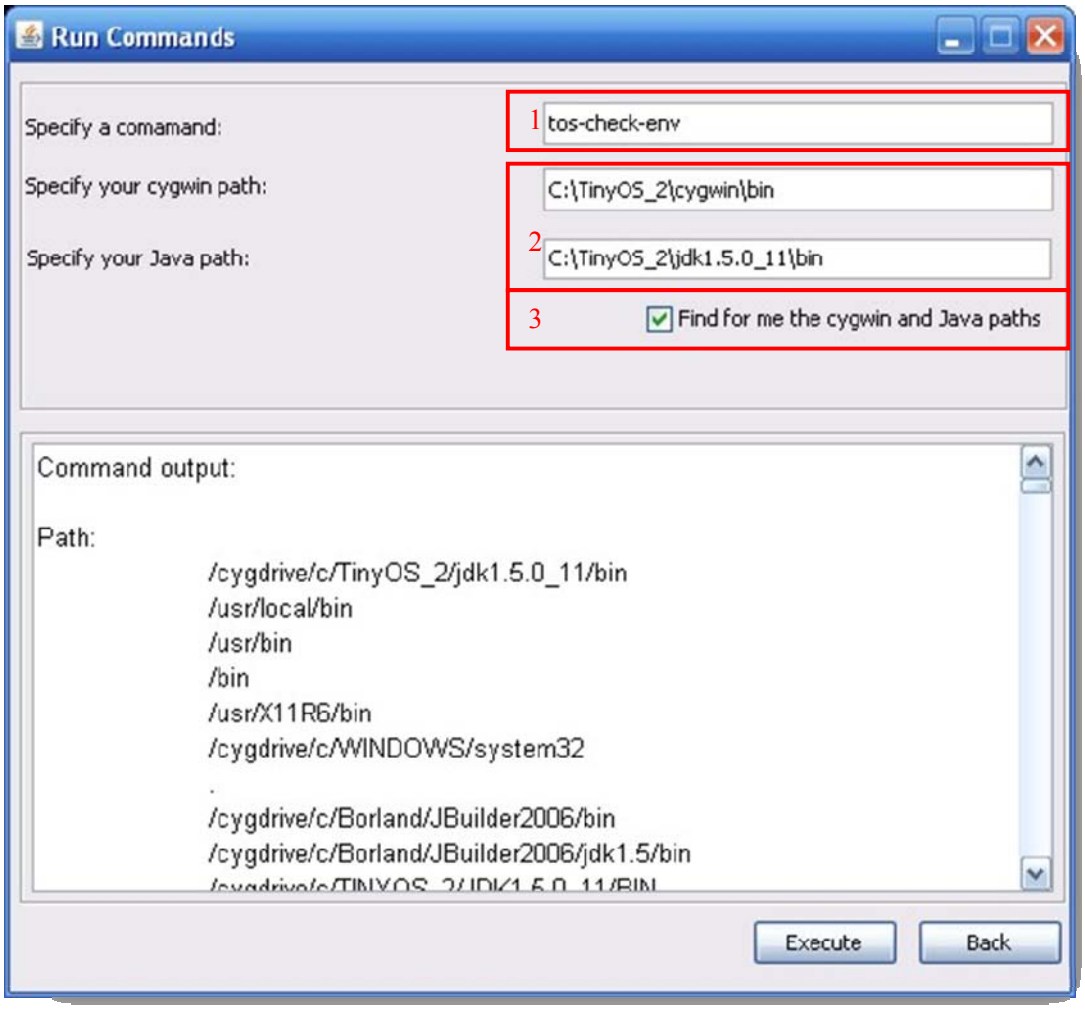
D3.1. Run a Command

Using WiSecurity you can easily manage your WSN `s nodes at any time. You can specify any cygwin compatible command and run it, without using the black box environment of cygwin. In order to run a command, go to:

Tools → Manage Motes → Run Command



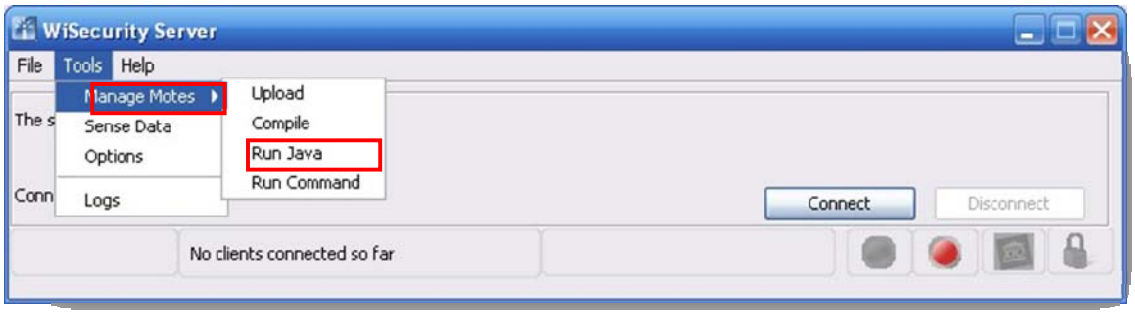
A new window will appear, specify a new command on the area one and the cygwin/ Java paths on area two. However, instead of adding manually the cygwin/ Java paths, WiSecurity could find it for you by checking the "Find for me cygwin and Java paths" checkbox on area three.



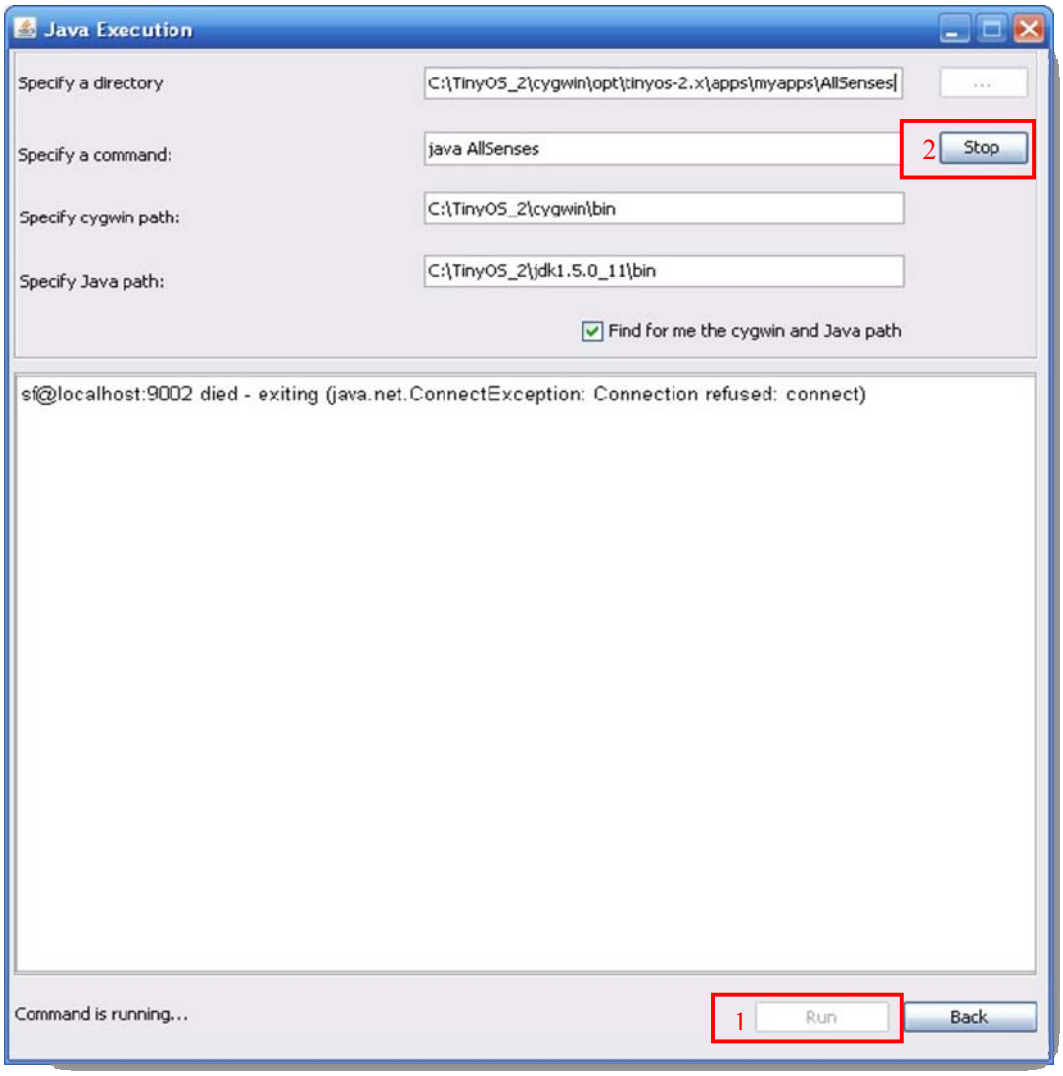
D3.2. Run a Java Application

You can run, in a real time way, using a cygwin – based format any Java applications from WiSecurity. Go to:

Tools → Manage Motes → Run Java



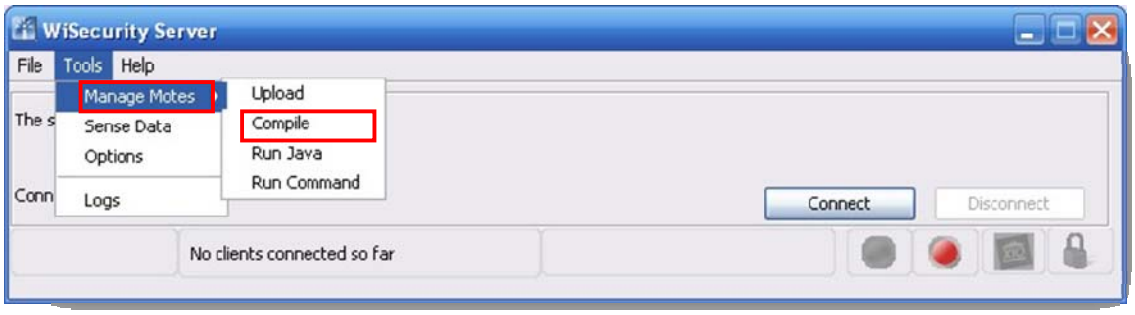
A new window will appear, specify the path where the folder with the Java application is located on your local disk. Next specify your command and the cygwin/ Java paths. However, instead of adding manually the cygwin/ Java paths, WiSecurity could find it for you by checking the "Find for me cygwin and Java paths" checkbox. At the end, click the "Run" button, on area one. At any time you can abort the execution of your Java application by clicking the Stop button on area two.



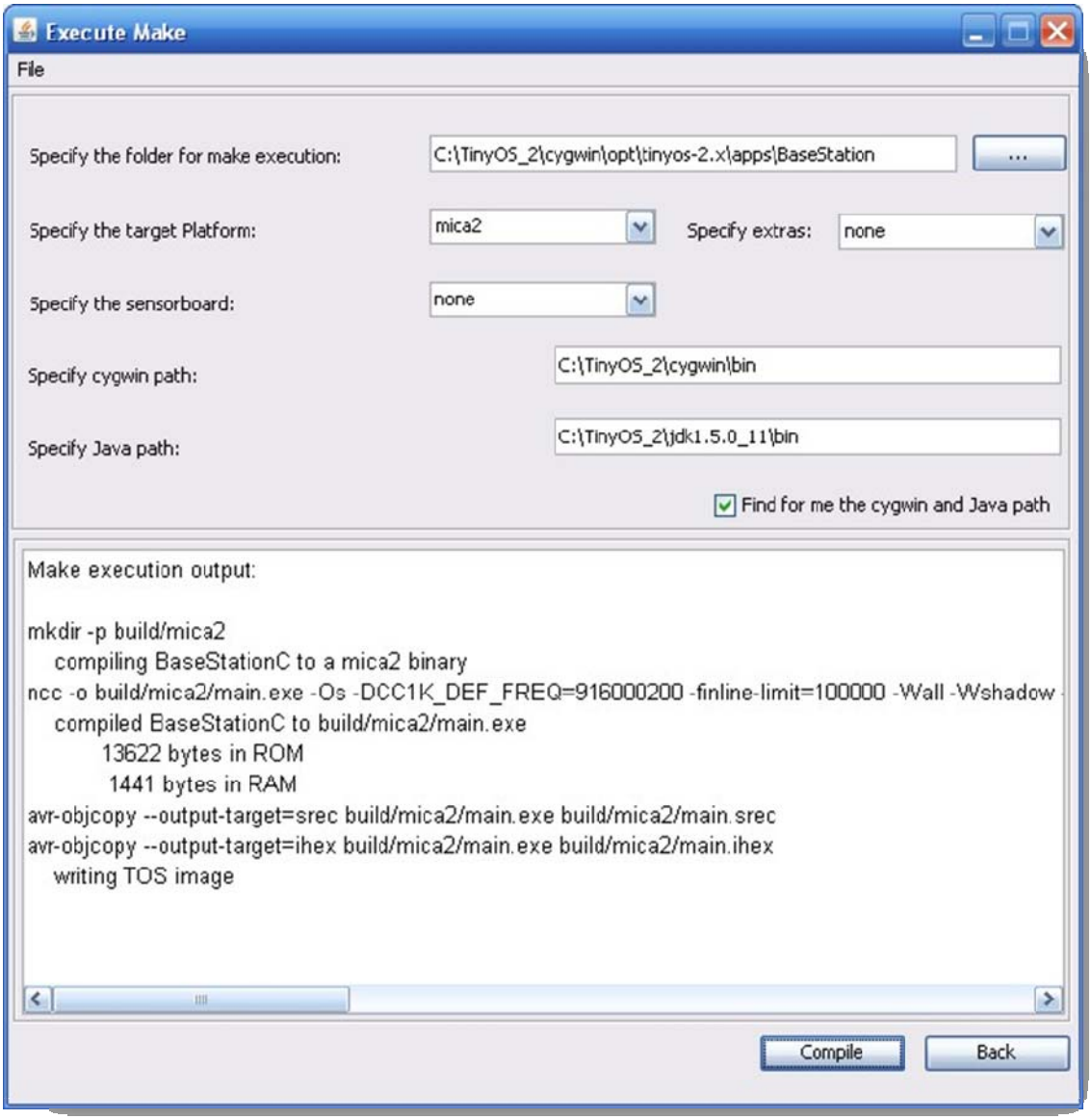
D3.3. Compile Code

WiSecurity is able to compile any nesC – based application using the TinyOS 2.0 configuration files. Go to:

Tools → Manage Motes → Compile



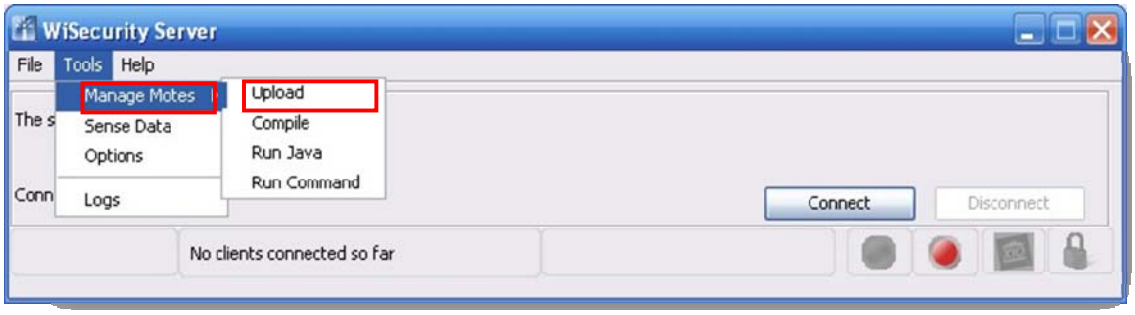
A new window will appear, specify the path where the folder with the compiled or non – compiled application is located on your local disk. Next specify the target platform, the sensor board model, any compilation extras and the cygwin/ Java paths. However, instead of adding manually the cygwin/ Java path, WiSecurity could find it for you by checking the "Find for me cygwin and Java paths" checkbox. At the end, click the "Compile" button.


Execute Make
File
Specify the folder for make execution:
C:\TinyOS_2\cygwin\opt\tinyos-2.x\apps\BaseStation
...
Specify the target Platform:
mica2
Specify extras:
none
Specify the sensorboard:
none
Specify cygwin path:
C:\TinyOS_2\cygwin\bin
Specify Java path:
C:\TinyOS_2\jdk1.5.0_11\bin
Find for me the cygwin and Java path
Make execution output:
mkdir -p build/mica2
compiling BaseStationC to a mica2 binary
ncc -o build/mica2/main.exe -Os -DCC1K_DEF_FREQ=916000200 -finline-limit=100000 -Wall -Wshadow
compiled BaseStationC to build/mica2/main.exe
13622 bytes in ROM
1441 bytes in RAM
avr-objcopy --output-target=srec build/mica2/main.exe build/mica2/main.srec
avr-objcopy --output-target=ihex build/mica2/main.exe build/mica2/main.ihex
writing TOS image
Compile
Back

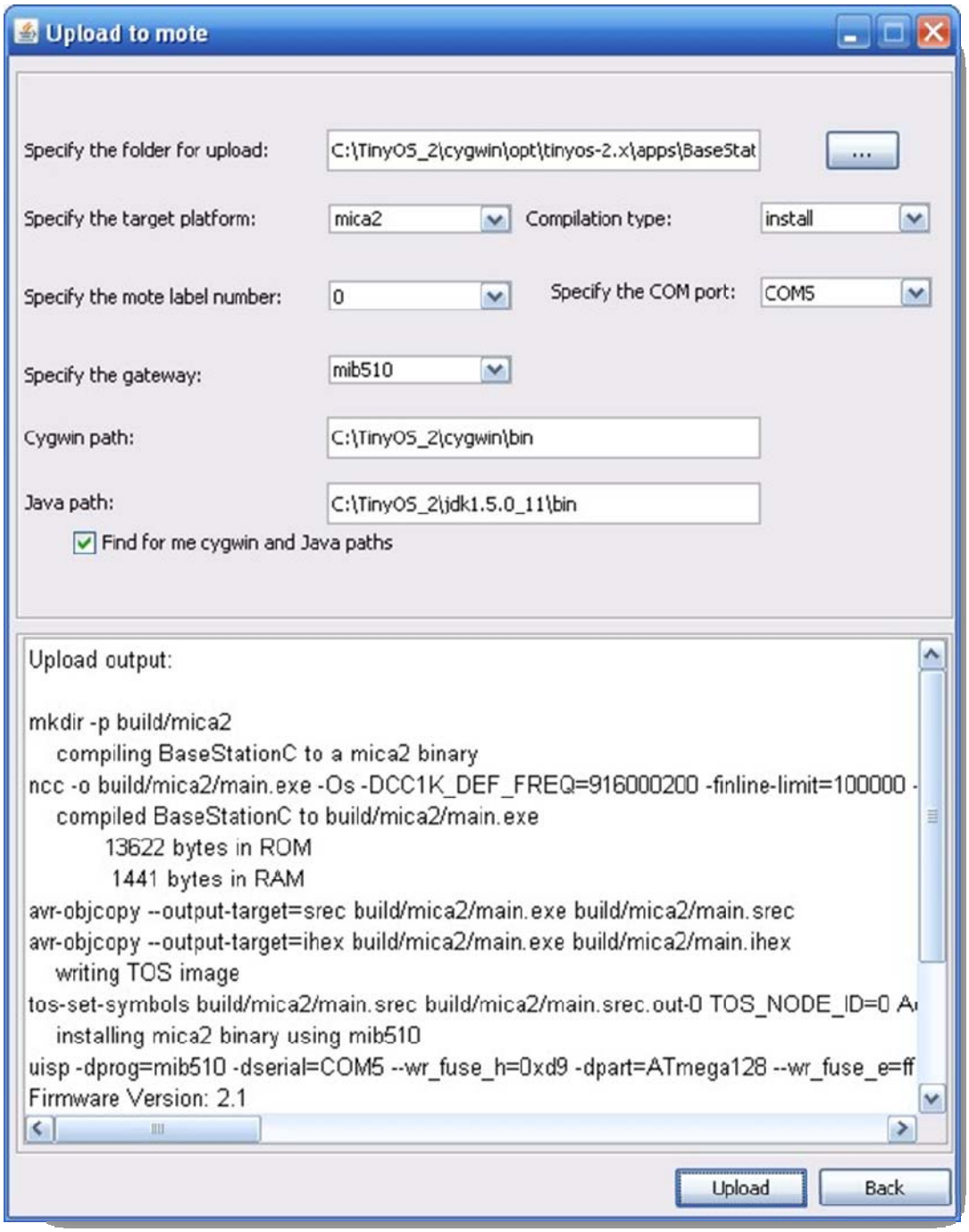
D3.4. Upload an Application

With WiSecurity you are able to upload an application using the TinyOS 2.0 configuration files on a node. Go to:

Tools → Manage Motes → Upload



A new window will appear where you can specify the folder on your local disk with the application you want to upload, the platform, the ID of the target node, the interface board that you use and the communication port where interface board is attached. In addition, you can compile a nesC – code and then upload to the target node (by selecting the install operation) or just upload an already compiled application (by selecting the reinstall operation). Finally, you have to specify the cygwin/ Java paths. However, instead of adding manually the cygwin/ Java path, WiSecurity could find it for you by checking the "Find for me cygwin and Java paths" checkbox.


Upload to mote
Specify the folder for upload:
C:\TinyOS_2\cygwin\opt\tinyos-2.x\apps\BaseStat
...
Specify the target platform:
mica2
Compilation type:
install
Specify the mote label number:
0
Specify the COM port:
COM5
Specify the gateway:
mib510
Cygwin path:
C:\TinyOS_2\cygwin\bin
Java path:
C:\TinyOS_2\jdk1.5.0_11\bin
Find for me cygwin and Java paths
Upload output:
mkdir -p build/mica2
compiling BaseStationC to a mica2 binary
ncc -o build/mica2/main.exe -Os -DCC1K_DEF_FREQ=916000200 -finline-limit=100000 -
compiled BaseStationC to build/mica2/main.exe
13622 bytes in ROM
1441 bytes in RAM
avr-objcopy --output-target=srec build/mica2/main.exe build/mica2/main.srec
avr-objcopy --output-target=ihex build/mica2/main.exe build/mica2/main.ihex
writing TOS image
tos-set-symbols build/mica2/main.srec build/mica2/main.srec.out-0 TOS_NODE_ID=0 A
installing mica2 binary using mib510
uisp -dprog=mib510 -dserial=COM5 --wr_fuse_h=0xd9 -dpart=ATmega128 --wr_fuse_e=ff
Firmware Version: 2.1
Upload
Back

D4. How to use the WiSecurity client

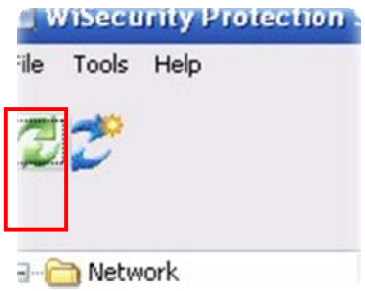
This chapter will guide you on how to use the WiSecurity client. How to make a connection with the server, how to manage the alarm and fire protection system and the x10 – based devices.

D4.1. Make a Connection

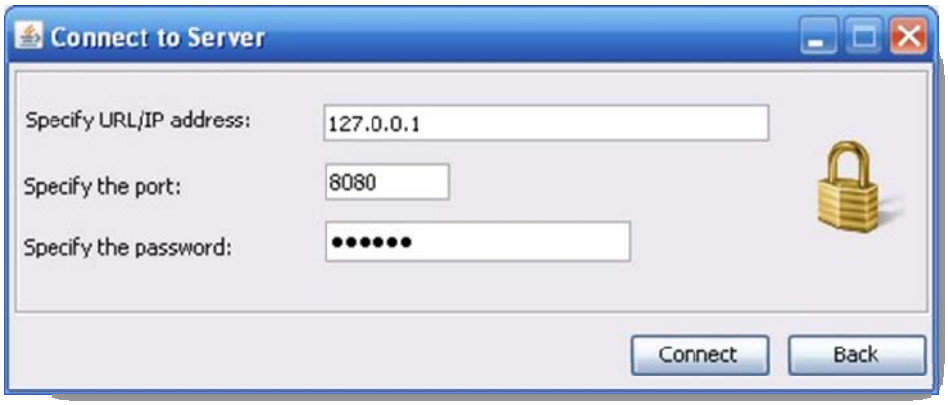
For making a new connection with the WiSecurity server, go to:

File → Connect, or

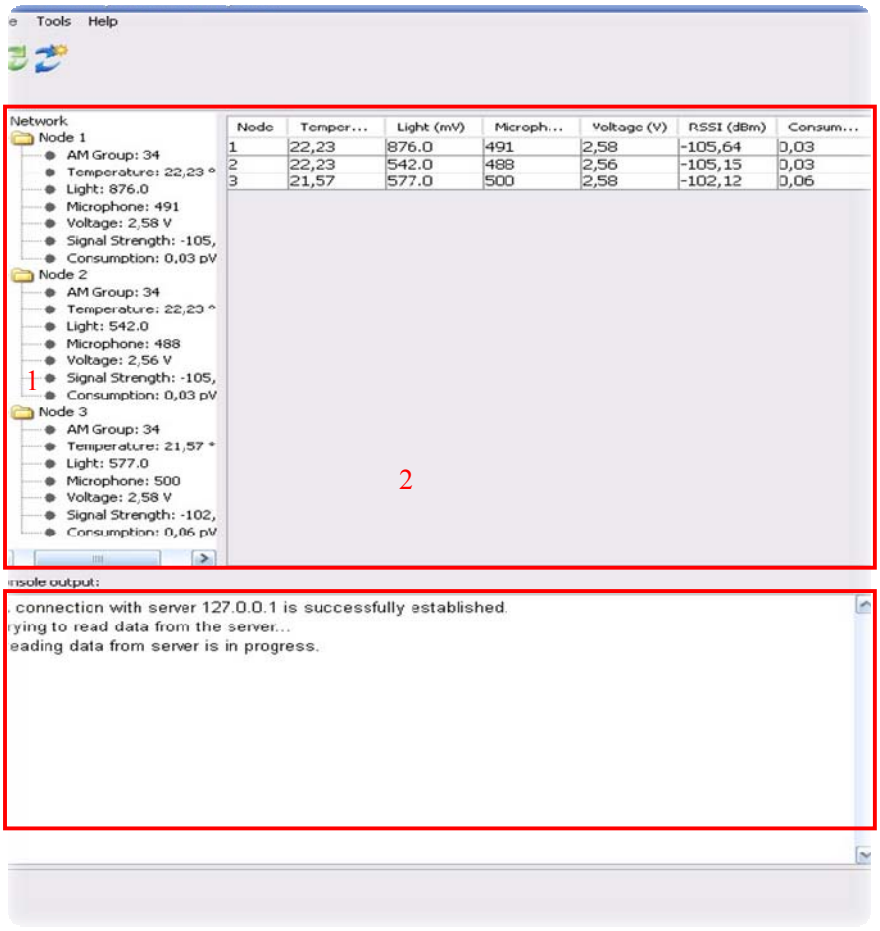
Use the icon the upper left corner of the screen (area one).



A new window will appear, like the picture below. Here you have to specify the IP location and the listening port of the WiSecurity server. Additionally, you have to add the secret password for the SSL communication between you (the client) and the WiSecurity server.

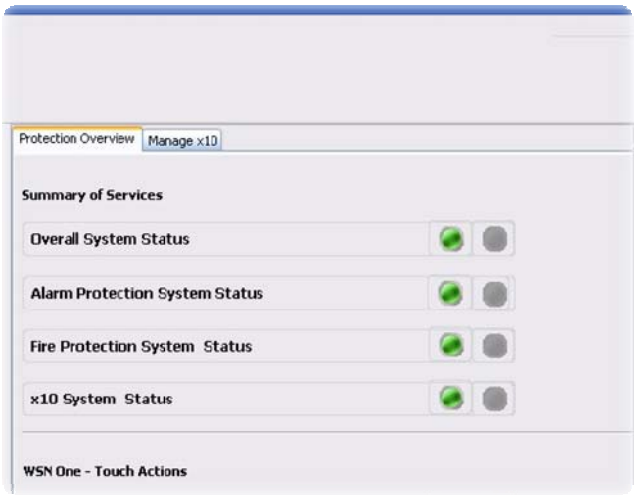


If a connection successfully established, you should see a message on area one indicating that the system is trying to read data from the server. After a while area two will starts to display the data coming from the WSN and in turn from the WiSecurity Server.



D4.2. Manage the Security and Fire Protection systems

As far as, you successfully connected with the server, you are able to manage the WSN, the security and fire protection systems. Using a series of icons you can read, at any time, the current status of the whole system. As you can see on the picture below, with a glance you can observe the overall system status, the alarm protection system status, the fire protection system status and the x10 network status. All of this information is received from the server, during your first connection and is updated any time you change the system status.

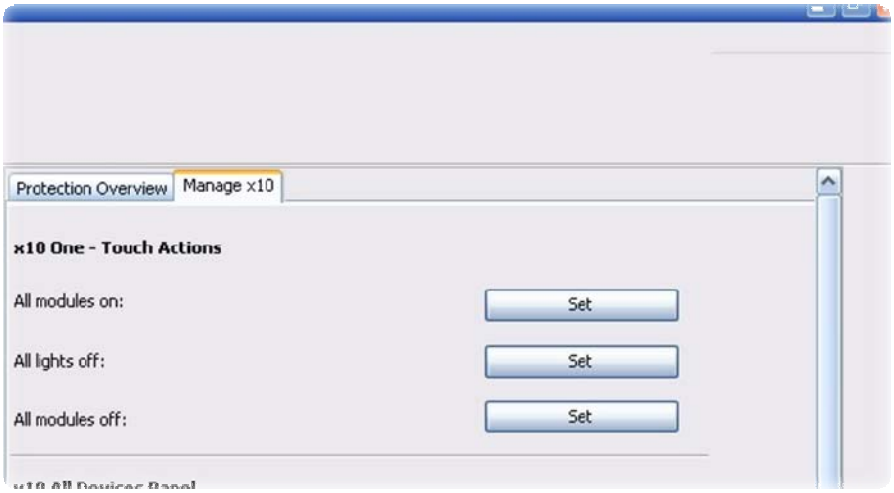


Using a collection of buttons, as the picture below shows, you are able to enable/ disable the Security or Fire Protection systems, from any place on the world. In addition, you can activate the alarm or the sprinkler devices, any time you need to do so.

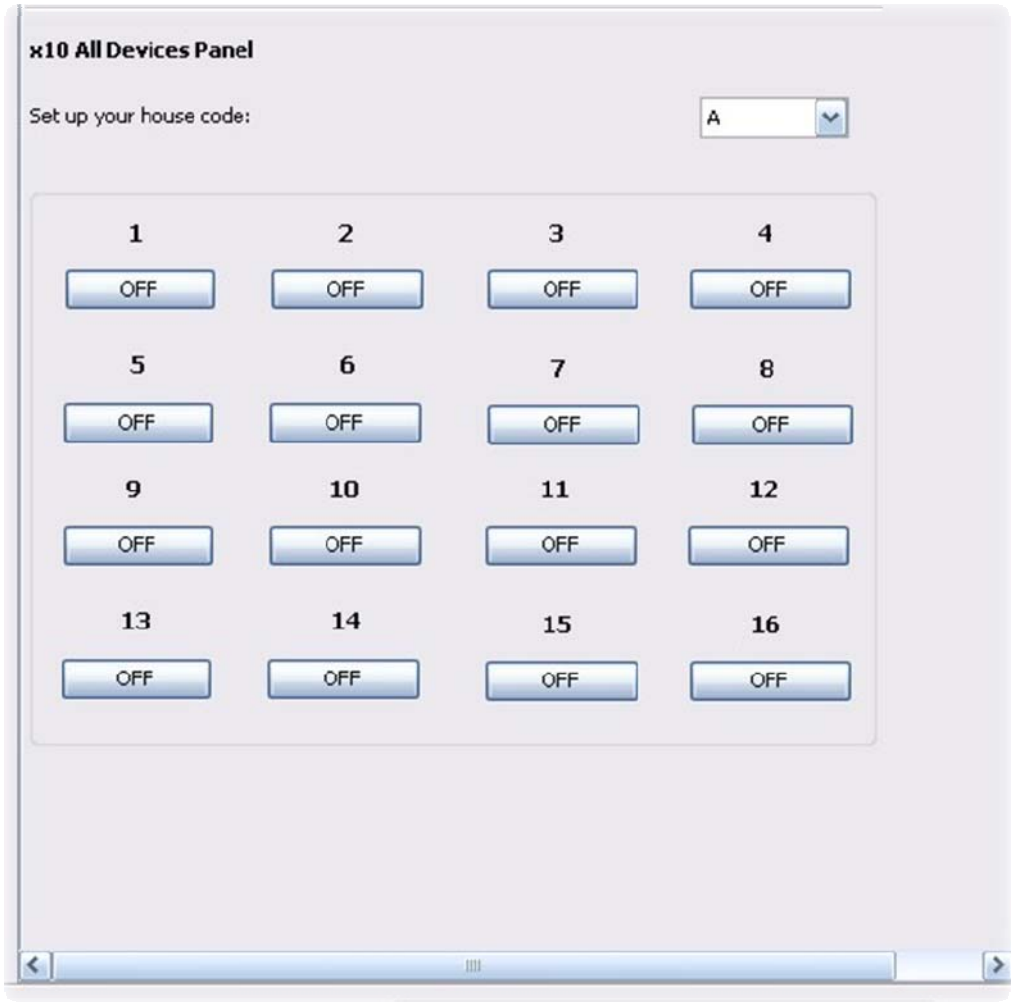
D4.3. Manage the x10 Network

WiSecurity client application, offers a complete x10 control – based interface for managing your x10 network remotely. You can manage your network at the same time you read data from the WSN or controlling the Security and Fire Protection system.

As you can see on the picture below, with the x10 control you are able to open or close all modules (including light and appliance devices) or close all the light modules.



In addition, you can manage each node separately, using a collection of sixteen toggles. You can specify the address code and code number of each device for turning it on or off.



D4.4. Manage the WSN

WiSecurity client enables the remote management of the WSN. As you can see on the picture below, you are able to set the nodes` leds status to on or off mode or change the time interval between the sensing periods for each node on the WSN. When, you first connect with the WiSecurity server both the status of leds and interval, are updated to the current settings from the WiSecurity server.